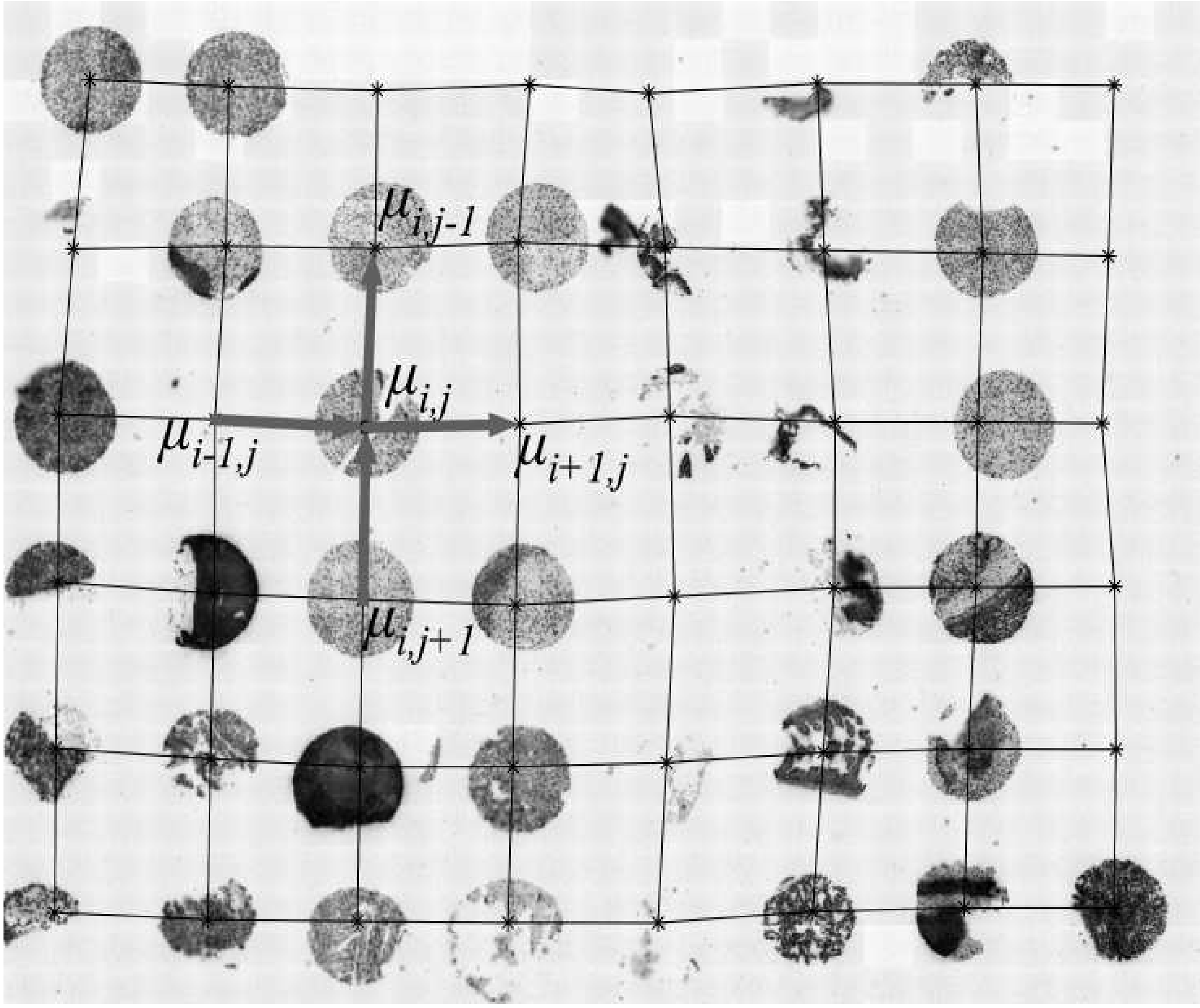
Eighth Workshop on Mining Scientific and Engineering Datasets (MSD'05) *

April 23, 2005

Sutton Place Hotel
Newport Beach, CA



Organizers:

Michael C. Burl, Jet Propulsion Laboratory
Arnold Goodman, University of California, Irvine
Chandrika Kamath, Lawrence Livermore National Laboratory

*** Held in conjunction with the Fifth SIAM Conference on Data Mining**

Foreword

There is a significant amount of innovative data mining work taking place in the context of scientific and engineering applications that is not well-represented in the mainstream KDD/Data Mining conferences. For example, scientific data mining techniques have been developed and applied to diverse fields such as remote sensing, physics, chemistry, biology, astronomy, etc. In these areas, data mining frequently complements and enhances existing analysis methods based on statistics and exploratory data analysis.

On the surface, it may appear that data from one scientific field, say genomics, is completely different from data in another field, such as physics. Nevertheless, there is much that is common across the mining of different scientific datasets. For example, techniques used to identify objects in images are very similar, regardless of whether the images came from a remote sensing application, a physics experiment, astronomy observations, or a medical study.

New techniques for scientific experimentation and data collection are continually being developed and utilized. As one example, biology laboratories are increasingly using robots to drive experiments that yield extensive amounts of microscopy data per night. In other disciplines, large-scale numerical simulations are being used to investigate physical phenomena that would be infeasible to study in the laboratory. Such simulations produce vast quantities of data often in the form of meshes or particle trajectories. Data mining techniques will clearly be indispensable for helping to transform these and other datasets generated by new experimental methods into information the scientist or engineer can use.

This workshop, which is the eighth in the series, is focused on bringing together data miners and scientists/engineers who are analyzing datasets from diverse fields to:

- share their experiences with other researchers working with similar data
- learn how methods developed for other disciplines might apply to their own data
- better understand new analysis techniques that are being developed in the scientific data mining community

As a final note, the scope of the workshop this year was expanded to include all aspects of bioinformatics. The growing interest in this area is well-reflected in the Workshop through one of the keynote talks and several of the contributed papers. We hope that you find the workshop interesting and enjoyable!

Sincerely,

Michael C. Burl, Jet Propulsion Laboratory
Arnold Goodman, University of California, Irvine
Chandrika Kamath, Lawrence Livermore National Laboratory

Program Committee:

Serge Belongie, University of California, San Diego
Shawn Newsam, Lawrence Livermore National Laboratory
Rahul Ramachandran, University of Alabama, Huntsville
Jelena Tesic, IBM TJ Watson Research Center

Additional Reviewers:

Kristin Branson
Piotr Dollar
Xiang Li
Sunil Movva

Acknowledgment:

The cover illustration was provided by **Joseph Roden** of the Jet Propulsion Laboratory.

Caption: Constrained clustering and lattice-fitting are used iteratively to identify tissue core locations and measure fluorescence signals within the core samples. The cover illustration shows a tissue array and the location of the lattice after the first iteration.

Workshop Website:

`http://www-aig.jpl.nasa.gov/MSD/`

Schedule

08:15–08:30	**Opening Remarks**
08:45–10:00	**Morning Keynote** *Collaborative Problem Solving: Link Between the Data and Satisfied Clients*, Arnold Goodman
10:00–10:30	**Break**
10:30–10:55	*Enabling Automated Analysis of Fluorescent In-Situ Hybridized Tissue Arrays Using Image Analysis with Constrained Clustering*, Joseph Roden, Lucas Scharenbroich, Victoria Gor
10:55–11:20	*Cloud Detection from Satellite Imagery: A Comparison of Expert-Generated and Automatically-Generated Decision Trees*, Smadar Shiffman
11:20–11:45	*A MISR Cloud-type Classifier Using Reduced Support Vector Machines*, Dominic Mazzoni, Akos Horvath, Michael Garay, Benyang Tang, Roger Davies
11:45–12:10	*Canonical Correlation, An Approximation, and the Prediction of Protein Abundance*, Anthony Bonner, Han Liu
12:10–01:30	**Lunch** (attendees on their own)
01:30–02:45	**Afternoon Keynote** *Image understanding and modeling for biological development: the case of a plant shoot meristem*, Eric Mjolsness
02:45–03:15	**Break**
03:15–03:40	*LibFeature: A Software Library for Quickly Generating Feature Vectors on the Fly from Structured Data*, Dominic Mazzoni
03:40–04:05	*A Data Mining Approach to Matrix Preconditioning Problem*, Shuting Xu, Jun Zhang
04:05–04:30	*Better Bond Angles in the Protein Data Bank*, C.J. Robinson, D.B. Skillicorn
04:30–04:35	**Closing Remarks**

Morning Keynote

Collaborative Problem Solving: Link Between the Data and Satisfied Clients

Arnold Goodman, Ph. D.
University of California, Irvine

Abstract

Today's science and engineering are driven by complex multidisciplinary problems that need the integration of data mining and statistics in successful solutions to satisfy clients. Such solutions should not begin before agreeing on a definition of the client's problem, and should not end before adding value with insights to facilitate his decisions or actions.

Complex multidisciplinary problems also require project teams to collaborate. In turn, genuine collaboration demands commitment to the client and his team, communication with them, and evaluation outside methodology and technology in client's environment.

Although collaborative problem solving appears fundamental, it is currently conspicuous by its absence. Checklists and scorecards of "what to do" and "how to do it" are covered, as a guide to collaborative problem solving.

Biography

Arnold Goodman has 45 years of experience involving statistics and information technology in the aerospace, petroleum, government, and university settings. He co-founded symposia on the interface of computing science and statistics in 1967, which are ongoing to this day. He organized the first meeting on measurement of computer systems at the 1972 Fall Joint Computer Conference. He was elected a fellow of the American Statistical Association in 1974. He co-founded the UCI Center for Statistical Consulting in 1997. He has also co-founded workshops on critical success factors for consulting and collaborating. His current work focuses on collaboration in data mining and software engineering and uncertainty in proteomics.

Afternoon Keynote

Image understanding and modeling for biological development: the case of a plant shoot meristem

Professor Eric Mjolsness
Institute for Genomics and Bioinformatics, and
Department of Computer Science
University of California, Irvine

Abstract

The Computable Plant project is a systematic effort to advance the understanding of the shoot apical meristem (SAM) of Arabidopsis thaliana through imaging and computational modeling of developmental processes. Interesting and generic problems arise within this computational approach. For example, to quantify the growth of the SAM and its cell lineages requires detecting and tracking multiple features in 3D image sequences, and finding a smoothed global velocity field due to growth; we approach this problem through nonlinear optimization. Also, fitting the resulting data to dynamical models requires a flexible modeling framework for coupled mechanical and regulatory networks. For these problems we develop a mathematical foundation based on the use of a ?dynamical grammar? capable of representing discrete-time events such as cell division that change the number of objects and their relationships, as well as continuous-time processes arising from regulatory networks and mechanical interactions. The resulting algorithms are being used to assist experimental research on mechanisms of meristem maintenance and phyllotaxis.

Joint work with Tigran Bacarian, Pierre Baldi, Ashish Bhan, Victoria Gor, Marcus Heisler, Henrik Jnsson, Elliot Meyerowitz, Venu Reddy, Alex Sadovsky, Bruce Shapiro. Further information available at `http://www.computableplant.org`.

Biography

Eric Mjolsness received an undergraduate degree from Washington University in St. Louis in 1980 and a PhD in physics from the California Institute of Technology in 1986. He has served on the faculties of Yale University, the University of California San Diego, the California Institute of Technology, and the University of California Irvine where he is currently a member of the Institute for Genomics and Bioinformatics and the Computer Science Department. He has also served as a leading member of the Machine Learning Systems Group at the Caltech Jet Propulsion Laboratory. His research interests are largely connected with the construction of scientific inference systems, using techniques from machine learning, pattern recognition, nonlinear optimization, statistical physics and other mathematical disciplines to further research into computational biology and other sciences.

Table of Contents

Enabling Automated Analysis of Fluorescent In-Situ Hybridized Tissue Arrays Using Image Analysis with Constrained Clustering

Joe Roden
Jet Propulsion Laboratory
MS 126-367
Pasadena, CA 91109
Joe.Roden@jpl.nasa.gov

Lucas Scharenbroich
School of Information and Comp. Sci.
University of California - Irvine
Irvine, CA 92712

Victoria Gor
Jet Propulsion Laboratory
MS 126-367
Pasadena, CA 91109

March 3, 2005

Abstract

The advent of high-throughput imaging techniques in biology in recent years necessitates reliable software techniques for automating data collection as much as possible. In this paper we describe a new software tool that enables automated analysis of tissue arrays, using an approach that combines traditional image processing techniques with a constrained clustering lattice-fitting technique to identify tissue core locations and measure fluorescence signals within the core samples. Our technique is able to robustly handle various distortions of the ideal grid as well as missing tissue cores. The technique is applied to tissue arrays with good results, enabling high-throughput collection of gene expression data from this useful biological assay.

1 Introduction

Scientist's efforts to decode the relationships between functional components of the genome rely on an increasingly diverse set of high-throughput experimental techniques to provide high-quality quantified data on a genome-wide scale. The workhorse of these techniques is the gene expression microarray, which typically measures relative quantities of thousands of transcripts in a single tissue sample. A complementary assay, fluorescent in-situ hybridization (FISH), seeks to quantify the degree to which one particular gene transcript occurs throughout an anatomical system. Hybridizing one cDNA probe to cross-sections of embryonic animals small enough to fit on a glass microscope slide can provide an anatomy-wide view of the gene's expression, but the resulting digitally-scanned images are not amenable to automated analysis at this time. Rather, an array of tissue section "cores" placed on a microscope slide is a recently developed technique that is particularly well suited to high-throughput screening and, to some degree, automated interpretation. Ongoing efforts to thoroughly annotate genes for completed genomes promise to benefit greatly from high-throughput custom FISH tissue arrays. This has lead us to develop tools to automate a substantial portion of the tissue array image data collection and analysis to support genome-wide annotations.

Our overarching goal is to provide a system to aid pathologists or researchers tasked with observing and annotating an individual gene's expression within an array of tissue cores. Researchers currently use a tissue microarray data collection system that requires them to review every tissue core within hundreds of digitally scanned tissue arrays and document the tissues in which a gene is expressed as well as their interpretation of the strength of the expression pattern (negative, uninterpretable, weak positive or strong positive) [4]. This existing system makes no attempt to automatically analyze or understand the image content, so it has no ability to direct the researcher's attention to a specific core location within an image. Instead that system's annotation process relies on the researcher to manually associate a core's address within the tissue array grid with that core's location within a tissue array scan.

As a component of our solution, we developed a tool to automatically analyze the digitally scanned tissue array images in order to locate the tissue cores and subsequently measure the strength of expression (relative fluorescence) within each of the tissue cores. Our system can use the resulting data to direct a researcher to a subset of tissues deemed to have significant expression, thereby skipping the interpretation of negative tissues and reducing the time required to review and annotate the tissue arrays. Further, our solution can further enable semi- or fully-automated microscopy by driving a microscope and/or scanning system to only take high-resolution images for these regions of interest.

The tissue array image analysis task is made challenging by numerous real-world factors related to the tissue array construction process: some of the tissue cores may not be present in certain slides; individual cores may be degraded so that only a portion of the core is present, or the core shape may be distorted or fragmented; the normally regular grid pattern of cores can be slightly rotated, warped or even torn; and the scans can have some unintended but visible artifacts or defects, e.g. spurious water spots. Additionally the tool was required to run relatively quickly – the system designers request that the program take no longer than a minute or so to locate and measure the cores present in the large scanned images.

Through a process of evolutionary design we constructed a package that robustly and efficiently performs job of locating and measuring the tissue cores, using a methodology that combines some aspects of the above existing techniques, e.g. adaptive segmentation, with some more novel techniques for spot finding. Our novel approach includes a fixed diameter matched filter approach to get an initial estimate for the placement of an idealized grid, in combination with a constrained-clustering-based adaptive shape segmentation that permits our grid to adapt to the actual data in the image while tolerating missing spots. In the following sections we review existing spot finding approaches, describe our software package and the details of our methodology, and follow with examples and results.

2 Spot Finding Approaches

Numerous image analysis approaches exist to locate and identify individual "spots" comprising an array, i.e. spots that are organized into rows and columns. It is worth mentioning that general image segmentation techniques alone or adaptive shape segmentation such as watershed [8][7] and seeded region growing [1] are inadequate because they fail to identify missing spots, may not handle artifacts, may not handle tissue cores that are fragmented, and don't solve the tissue core grid address to pixel location mapping problem per se. We believe this problem requires an approach that takes full advantage of the known lattice or grid-based structure of the data, in particular the grid size and spacing and core sizes.

Yang et. al. [11] reviewed microarray spot finding and measuring techniques, and described the three phases of the task: *addressing* or *gridding* to associate tissue array grid coordinates with pixel locations; *segmentation* to separate foreground and background pixels; and *intensity extraction* to measure fluorescence intensity. Within this framework many variations have been explored in an attempt to maximize the quality of the resulting fluorescence measurements, e.g. Yang et. al. 2002 [10], Wu and Yan 2003 [9], Park et. al. 2004 [6]. However all of these existing approaches are tailored to some degree to gene expression microarrays and so they do not necessarily generalize well to tissue arrays considering the variability present with tissue arrays, e.g. grid distortions, grid rotation, missing and degraded cores (see Figure 1). For example, both Jouenne [3] and Agulo and Serra [2] use horizontal and vertical projections (image row and column sums) to identify array rows and columns, but their approach (and others') assumes that there is effectively no rotation or skewing of the grid and spots across the image. Furthermore they exploit details of microarray slide printing, e.g. print-tip irregularities, that are not relevant to tissue arrays. Thus our work has primarily focused on developing a robust solution to the more challenging tissue array addressing problem.

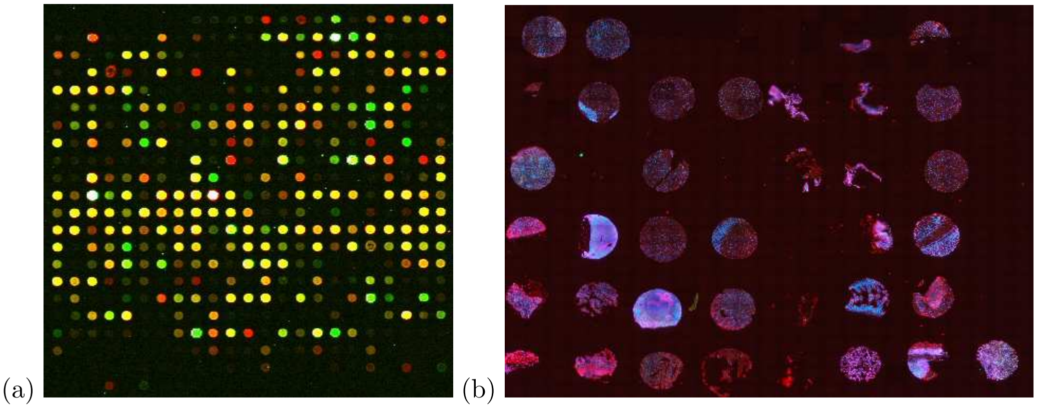


Figure 1: Examples of (a) a portion of a microarray scan and (b) a tissue array scan. Note that the tissue array exhibits greater grid distortion (including grid rotation, stretching and tearing), more missing spots, and spots with more diverse shape, internal structure and texture.

3 Software

We have created a software tool to analyze overview scans of tissue arrays to assist researcher's analysis efforts by focusing them on the small set of tissue cores that have the strongest expression signals. This tool has been integrated into a larger system that permits researchers to collect, annotate and analyze a large number of custom in-situ hybridization scans to support biologist's investigations. A pseudocode description of the spot finding and measuring software follows:

```
- locate tissue cores
  - initialization
    - compute response surface for core
      spot matched filter
    - search (exhaustively translate &
      rotate an idealized grid) to find
      grid position w/ maximum response
  - relax grid intersection points
    - k-means with 2D grid constraint
- measure each core's signal
  - sum of foreground pixel's red
    channel intensity
```

The program is written in C, with minimal dependencies on third-party software for maximum portability and ease of integration into the larger in-situ image management system. The software uses FFTW version 3 library to efficiently perform the filter convolutions, and uses the ImageMagick C++ API library for image reading and writing. The portions of the software written in-house total approximately 3400 lines of code. The larger in-situ management system has facilities for visualizing and adjusting the resulting core locations prior to the results interpretation phase.

For efficiency, the addressing or gridding portion of the software may optionally be performed using a downsampled version of the image. Upon completion the resulting core center locations can be scaled back up to the original image coordinate system. When extracting the fluorescence data, the original-resolution image should be analyzed in order to use the highest-quality information available for each of the tissues; speed is not as great a concern at this later stage of analysis.

4 Locating Tissue Cores

The tissue array is designed to contain a set of cores located on regular grid positions, e.g. an array might have 7 rows and 8 columns of tissue cores, generally of equal size and evenly spaced. The names of the tissues placed at each grid location are known in advance, but the software system has to derive the location within the scan of each core, and the extent of the core, so that it can accurately measure the fluorescence signal corresponding to each tissue. The solution we settled on combines the speed of image processing (to place an initial grid) with the flexibility of unsupervised clustering optimization (to relax the initial grid to tolerate some degree of distortion).

The initial grid placement phase searches exhaustively to find the best translation and rotation of an ideal grid. The approximate core locations that it identifies are then used as an initial seed to the relaxation phase. One cluster mean is defined for each possible core location (each initial grid intersecton point- the number of which is known a priori), even if the fluorescent signal is weak or non-existent in the immediate vicinity.

4.1 Placing the Initial Grid

In the given task, many attributes of the initial grid are known a priori. Such attributes are: number of rows R and columns C in the tissue grid, diameter of the tissue samples (the core diameter), and distance between cores. Nevertheless, automated analysis of fluorescent in-situ hybridized tissue arrays is complicated by the inaccuracies in the data. In addition to usual noise introduced by image generation process, we are confronted with various inaccuracies introduced by humans during the placement of delicate tissue sections within the grid of the microscopic slide. The human introduced errors include frequently missing cores, fragmented and malformed tissue samples, and inaccurate positions of the cores within the grid. In addition, the entire tissue array (grid) can be rotated and shifted while placed under the microscope.

Because of these inaccuracies the relaxation part of the software requires a good guess of where the original grid is positioned in the picture, that is it requires the knowledge of the shift and rotation undergone by tissue array. Once the position of the original grid is known, the relaxation algorithm deals well with finding exact core positions within the grid.

Luckily, the rotation and shift of tissue array introduced during slide placement under the microscope is not very large. The rotation of the image frame is limited to 25 degrees and shift is limited to 35 pixels. Therefore, it is possible to search exhaustively for best grid fit, given that the fit calculation for best grid is FAST.

To calculate the fit we use the ideal core as a matched filter and calculate filter response on original image, that is we perform correlation of filter $f(x, y)$ with image $g(x, y)$.

$$f(x, y) \bullet g(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) g(x+m, y+n)$$

where M and N are the width and height of the image in pixels. The ideal core $f(x, y)$ is represented by a white circle of the given core diameter on a black background (Figure 2.a.). The filter response $f(x, y) \bullet (x, y)$ (Figure 2.b.) is computed in frequency domain (correlation in the time domain is equivalent to multiplication in frequency domain). This enhances the efficiency of the algorithm. If the underlying image pattern is similar to the filter, the response image will have a high value at the given filter location, representing the goodness of individual core fitness at that location. We expect high responses in the vicinity of the regions where the actual cores are located, and low responses otherwise. The overall fitness of one possible grid can be calculated by summing the $R \times C$ individual core fitnesses, i.e. the filter response of the ideal core centered at each of the grid's $R \times C$ intersection points.

In an exhaustive search the overall grid fitness is calculated for each possible rotation and each possible shift. If we neglect the cost of calculating the grid intersection locations for each possible grid (e.g. if the coordinates of rotated and shifted grid intersections are calculated and presorted a priori), the overall grid fitness calculation can be performed as N additions, where N is the total number of grid intersection points (the number of cores). Such a calculation is extremely fast, enabling an exhaustive search for all possible grid locations, and producing an accurate estimate of grid position.

While the semi-regular placement of the tissue samples should result in a distinctive 2D discrete Fourier transform that indicates the frequency of the grid, this information is provided to our program in advance. The grid rotation and translation could also be recovered from the frequency image, but this would require similar image processing techniques to those we presently employ in the spatial image. We found our approach to be direct, efficient and effective, and so we chose not to analyze the frequency image.

4.2 Relaxing the Core Locations

The relaxation algorithm is a standard k-means clustering algorithm [5] augmented to constrain the relative positions of the cluster centroids.

In order utilize k-means, the image needs to be transformed into a "conventional" data set in feature vector (matrix) form. Pixels having a gray scale intensity that exceeds a user-controlled threshold are added to a two- dimensional dataset, X, of pixel instances, each with an x and y pixel location.

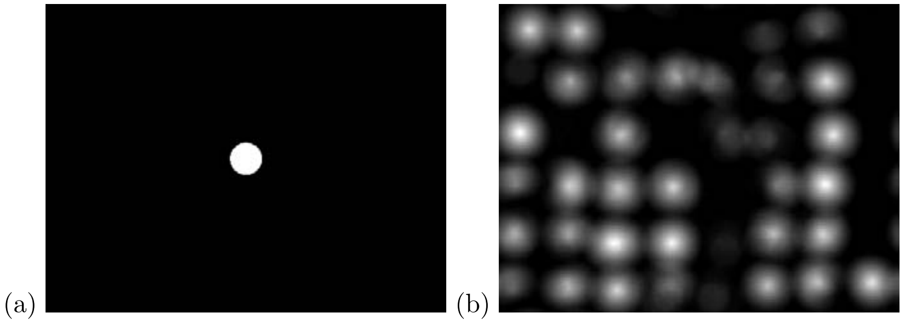


Figure 2: The (a) matched filter and (b) matched filter response surface that is used to compute the placement (offset and rotation) of an initial idealized grid.

During initialization, each of the initial cluster centroids is logically associated with a grid intersection point in the idealized grid produced by the previous stage (see Placing the Initial Grid). Since the number of rows, R, and columns, C, are known *a priori*, exactly $R \times C$ clusters are created with the first cluster mean mapped to grid point $(1, 1)$ and the last cluster mapped to grid point (R, C).

The algorithm iterates through the following three steps until convergence:

1. update means via standard k-means update
2. constrain the centroids using the current grid parameters
3. update the grid parameters

The grid parameters are represented by two vectors, $\boldsymbol{\Delta}_h$ and $\boldsymbol{\Delta}_v$, which correspond to horizontal and vertical unit steps, $(\Delta x, \Delta y)$, along the grid in pixel space. Since the grid may be skewed or rotated, these unit steps must be represented in vector form, rather than simple scalars.

4.2.1 k-means Update

The standard k-means update step recomputes the centroid of a cluster as the mean of all data points which lie closer to that particular centroid than any other. This is expressed as

$$\hat{\mu}_i^{(t+1)} = \frac{1}{n_i} \sum_{x \in X} x \; I(\mu_i^{(t)}, x)$$

where $I(\mu_i, x)$ is an indicator function which equals 1 if x is closer to μ_i than any other μ_j, $i \neq j$ and zero otherwise, and n_i is the number of points closest to μ_i, or $\sum_{x \in X} I(\mu_i, x)$.

In the degenerate case where a centroid has zero data points assigned to itself, the centroid's value remains unchanged from the previous iteration.

4.2.2 Constraining Centroids

Once the proposed centroids, $\hat{\mu}_i$, have been computed, they need to be forced to lie on the imposed grid. Since we only require that a loose grid structure be imposed, we constrain each centroid, i, in terms of its *neighborhood*, which is defined as i itself as well as the centroids at the grid positions which lie one unit away from i. These are the centroids immediately above, below, to the left and right of i.

$$\mu_i^{(t+1)} = \sum_{j \in \text{Neighborhood}} w_j (\hat{\mu}_i^{(t+1)} - \Delta_{ij}^{(t)})$$

where

$$w_j = \frac{n_j + 1}{N_i + |J|}$$

$|J|$ is the size of the neighborhood, which is at least 1 since i is a member of its own neighborhood, n_j is the count of the datapoints in neigboring cluster j, and N_i is the count of all the datapoints in the neighborhood of i. The offset between centroids Δ_{ij} is defined in terms of the grid offset parameters at the previous time step, $\boldsymbol{\Delta}_h^{(t)}$ and $\boldsymbol{\Delta}_v^{(t)}$.

This update equation can be interpreted as a weighted mean of the centroids where the weights correspond to the posterior mean of a multinomial distribution with a uniform Dirichlet prior. The multinomial coefficients represent the relative weights of the centroids in a given neighborhood and the observed data is composed of all the data points within the neighborhood of the i.th centroid.

4.2.3 Update Grid Parameters

The grid parameters, $\mathbf{\Delta}_h^{(t)}$ and $\mathbf{\Delta}_v^{(t)}$ are, intuitively, updated by computing the average horizontal and vertical displacement between constrained centroids $\mu_i^{(t+1)}$. These average displacements define a *grid basis* which may not be orthogonal. Since our idealized grid *is* orthogonal, we project the computed basis to enforce orthogonality. Finally, since the projection operation introduces a rotation in one of the basis vectors, the entire basis is rotated to compensate and the vectors are normalized to the proper length (see Figure 3).

For notational convenience, $\Delta_{h,x}$ represents the x component of the vector $\mathbf{\Delta}_h$ and $\mu_{i,j}$ refers to the centroid associated with the grid point (i,j).

First, the average displacements are computed,

$$\begin{aligned}\bar{\Delta}_h &= \frac{1}{R(C-1)}\sum_{i=1}^{R}\sum_{j=1}^{C-1}\mu_{i+1,j}^{(t+1)}-\mu_{i,j}^{(t+1)}\\ \bar{\Delta}_v &= \frac{1}{C(R-1)}\sum_{i=1}^{R-1}\sum_{j=1}^{C}\mu_{i,j+1}^{(t+1)}-\mu_{i,j}^{(t+1)}\end{aligned}$$

followed by a projection to enforce orthogonality of the basis vectors.

$$\bar{\Delta}_{v,y} = -\frac{\bar{\Delta}_{h,x}\bar{\Delta}_{v,x}}{\bar{\Delta}_{h,y}}$$

Finally, the induced rotation of the basis vector $\mathbf{\Delta}_v$ is computed and the entire basis is rotated by half this amount is order to evenly distribute the error between $\mathbf{\Delta}_h$ and $\mathbf{\Delta}_v$.

$$\begin{aligned}\theta &= \arccos\left(\frac{\bar{\mathbf{\Delta}}_h\cdot\bar{\mathbf{\Delta}}_v}{|\bar{\mathbf{\Delta}}_h||\bar{\mathbf{\Delta}}_v|}\right)\\ \mathbf{\Delta}_h^{(t+1)} &= \text{rotate}(\bar{\mathbf{\Delta}}_h,\frac{1}{2}\theta)\\ \mathbf{\Delta}_v^{(t+1)} &= \text{rotate}(\bar{\mathbf{\Delta}}_v,\frac{1}{2}\theta)\end{aligned}$$

These update equations are repeated until a convergence criterion has been reached, or a maximum number of iterations have been executed.

5 Measuring Fluorescence

The goals for measuring fluorescence for this effort were very modest. Unlike the microarray situation in which all expression quantification is automated, our scientists intend to continue to review and manually annotate cores showing a significant level of expression. Not only must the expert pathologist/researcher describe and record the pattern of expression observed in each tissue, but they must also describe in which subtissues and cell types the pattern of expression is observed. Thus it is not important that our software report highly-accurate levels of expression, since our goal is only to provide a means to rank order the tissues roughly by relative level of expression. In particular, the level of expression observed in one fluorescent channel, the red channel of our images, is of interest to our investigators.

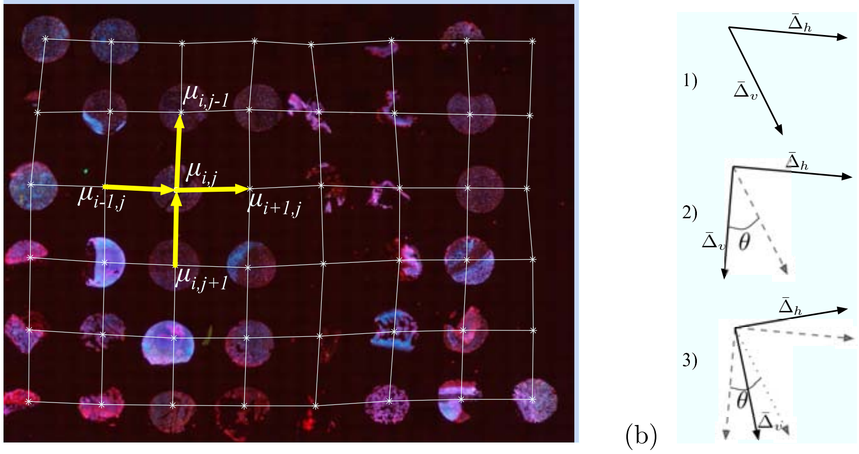
To that end we collect some basic statistical measurements for each tissue core's image region as identified by the prior tissue locating stage. Each tissue region is defined to have a center location as determined by the relaxed cluster centers, and all cores have the same fixed diameter based on the known core size (so that statistics are collected over an equal-sized area for each core). The statistics collected for each tissue include:

total red the sum of the red intensity of all pixels within the tissue

red area the total number of pixels within the core that exhibit sufficiently strong red channel intensity (over a red threshold predetermined by the investigators in prior studies)

In practice these two measures are quite well correlated, and either will suffice to provide a meaningful ordering of tissues that will help the researcher prioritize and streamline his annotation effort. Figure 4 shows the relative red scoring of three distinct tissues.

It is noteworthy that at this stage our approach does not involve foreground versus background separation via thresholding or adaptive shape segmentation. Tissue arrays exhibit much greater diversity of signal within their cores, reflecting diverse underlying tissues and cell types, and so it is not necessarily appropriate to partition a tissue core's region into two classes. Also a background pixel's contribution to the red channel scores we derive is negligible and so does not affect the resulting tissue prioritization.



(a) (b)

Figure 3: The cluster centers and connecting grid after one iteration. The grid parameters are derived by (a) averaging the neighborhood offsets (shown in yellow) from the cluster centroid $\mu_{i,j}$ across all grid intersection points (i,j) to obtain the intermediate variables $\bar{\mathbf{\Delta}}_h$ and $\bar{\mathbf{\Delta}}_v$, then by (b) rotating these basis vectors by $\frac{1}{2}\theta$ (to enforce orthogonality) to get the final grid parameters $\mathbf{\Delta}_h$ and $\mathbf{\Delta}_v$

.

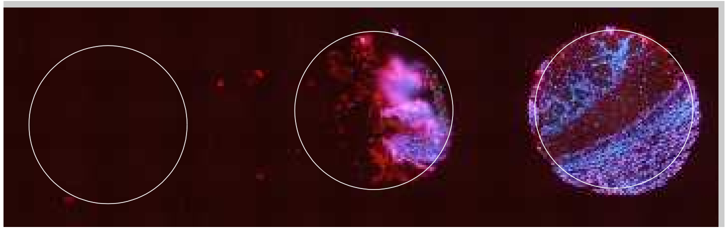


Figure 4: Relative red-channel fluorescence is estimated for each tissue. These three tissues have the following *red area* red pixel counts (from left to right): 0, 1727, and 1460. On this basis, the middle tissue would receive the highest priority for annotation of the three.

6 Results

A total of 20 tissue array images were provided for algorithm development and testing, all roughly in the 1000x1000 size range, composed of either 48 cores in a 6 by 8 grid or 56 cores arranged in a 7 by 8 grid. In this section we demonstrate the performance of the approach on one typical tissue array, that of Figure 1. Due to space constraints we will only discuss generally how our method performed on the larger set of tissue array images available to us during development.

In our testing we manually derived the approximate core size and spacing, as well as the appropriate threshold settings for foreground/background pixel separation. Very little adjustment is required to achieve the correct results, and most are known *a priori* from the experiment design. The input parameters used to obtain the results we show for this example are:

```
rows 6
cols 8
spot_delta 120
spot_diameter 80
max_iterations 100
threshold 0.12
```

Figure 5 shows the initial and final core locations computed for the tissue array in Figure 1. Figure 6 demonstrates that the localized tissues can be further analyzed for intensity of expression, and on that basis the tissues can be prioritized for manual annotation in order to maximize a researcher's efficiency.

We observe that the method does a good job of finding an appropriate initial guess for the grid, and that the core center relaxation adjusted well to settle on all the complete or partial core in the image. In even does quite well on the columns that exhibit a reasonable amount of skew, e.g. the second column from the right. The constrained clustering approach does a good job of pushing the tissue centers for the last, mostly empty column, off to the right away from the second, allowing the strong signals present in the second to last column to influence the shape of the grid in that portion of the image.

The software performed equally well on the remaining tissue array images used during development. Each of these early tissue arrays exhibited interesting forms of variation (some had more rotation, some had more skew or missing cores). In practice the users can provide a somewhat inaccurate grid spacing parameter, either slightly larger or slightly smaller than ideal, and in either case the software was able to settle to the correct grid. In a few cases cores with unusually- shaped cores resulted in center estimates that were slightly offset from ideal. Such a core's computed center plus radius was not positioned well enough to capture all of the relevant pixels, though they typically contain the majority of pixels. But in no case did this compromise the overall grid assignments or tissue prioritization.

For the example we illustrated, a slide overview scan of 1000x768 pixels, the tool calculated the final grid locations and measured the fluorescence in approx 30 seconds on an Apple Powerbook 800MHz G4 laptop computer with 1GB RAM. The time to converge in the relaxation step causes the greatest variation in run times, and poorly specified parameters (e.g. incorrect grid spacing or core diameter) can result in many more iterations, slow run times, e.g. 1-2 minutes, and incorrect grid assignment results.

7 Conclusions

When we completed the design of our matched filter grid estimation phase we considered an alternative approach, to use seeded region growing or its variants as a means to relax away from the ideal but inaccurate grid points to locate the core centers. We recognized early that for tissue arrays such an approach still requires some way to build in spatial constraints necessary to prevent neighboring cores from moving towards (and conflicting over) the same core, especially in the case of an adjacent empty core. Our constrained-clustering-based approach offered a much more elegant way to enforce those priors than the alternative: to manually graft directional constraints into the seeded region growing to make it fully respect the grid constraints.

Since our technique is applicable to any grid image, we believe it can be applied to microarray imagery as well as tissue arrays. We also intend to reformulate our relaxation algorithm within a probabilistic framework in order to guarantee convergence and more easily introduce prior knowledge from the filter matching.

The software has proven very effective at robustly identifying tissue cores within tissue arrays and prioritizing them for subsequent man-

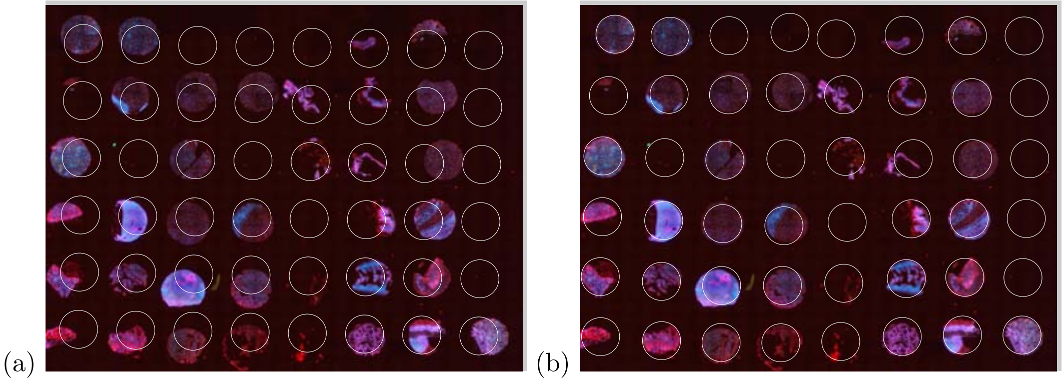


Figure 5: The initial idealized grid placement (a), and the final grid placement (b) after the constrained-clustering relaxation (6 iterations).

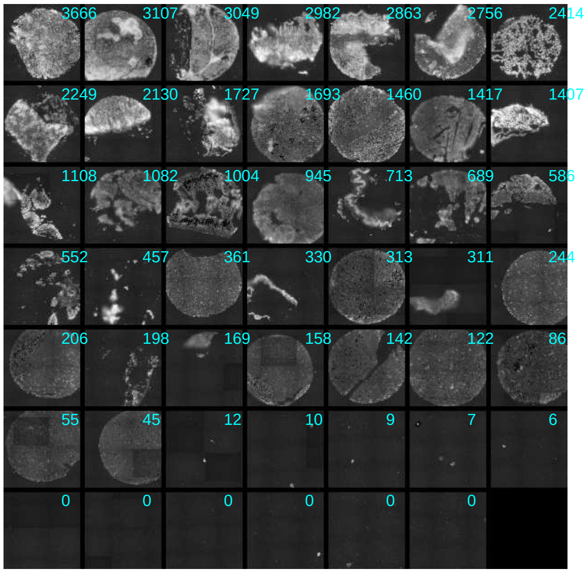


Figure 6: The red-channel fluorescence estimates (e.g. the *red area* values shown for each core above) can be used to prioritize tissues for subsequent annotation, and thereby speed up the data collection.

ual annotation. As we get more familiar with the complete image analysis and interpretation process and discover appropriate methods, we are hopeful that much of the burden of manual annotation and curation of tissue array expression data can be further reduced. It is reasonable to consider whether and to what degree we can employ learning algorithms to automate the more objective portions of the tissue expression annotation process. However, the pathologist/researcher often brings substantial expertise and knowledge of detailed anatomical structures to the process, and we cannot realistically hope to automate the experience-based recognition aspects of the process for the foreseeable future.

Finally, our success at using constrained clustering to bring to bear the right level of prior information and structure to solve this particular image analysis problem gives us confidence that there are more well-structured or semi-structured segmentation problems that can be addressed in a similar fashion. At least within biology many scientists can benefit from better tools and techniques to aid or enable various high-throughput assays, e.g. the diverse investigations that automated fluorescence microscopy makes possible.

Acknowledgments

We would like to thank Xiaochen Lu and Eddie Wehri at LLNL for generating and providing the images, as well as motive and assistance in developing this tool. We thank Barbara Wold and Lisa Stubbs for instigating and overseeing this research, and Brian Williams for helpful feedback. This work was supported by grants from the Department of Energy Office of Biological and Environmental Research.

References

[1] Adams, R., and Bischof, L. Seeded region growing. *IEEE Trans. Pattern Anal. Mach. Intell. 16*, 6 (1994), 641–647.

[2] Angulo, J., and Serra, J. Automatic analysis of DNA microarray images using mathematical morphology. *Bioinformatics 19*, 5 (Mar 2003), 553–562.

[3] Jouenne, V. Y. Critical issues in the processing of cdna microarray images. Master's thesis, Virginia Polytechnic Institute and State University, 2001.

[4] Liu, C. L., Prapong, W., Natkunam, Y., Alizadeh, A., Montgomery, K., Gilks, C. B., and van de Rijn, M. Software tools for high-throughput analysis and archiving of immunohistochemistry staining data obtained with tissue microarrays. *Am J Pathol 161*, 5 (Nov 2002), 1557–1565.

[5] MacQueen, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Symposium on Math, Statistics, and Probability* (1967), University of California Press, pp. 281–297.

[6] Park, C.-B., Lee, K.-W., and Lee, S.-W. Automatic microarray image segmentation based on watershed transformation. In *ICPR (3)* (2004), pp. 786–789.

[7] Roerdink, and Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *FUNDINF: Fundamenta Informatica 41* (2000).

[8] Vincent, L., and Soille, P. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE PAMI, 1991 13*, 6 (1991), 583–598.

[9] Wu, S., and Yan, H. Microarray image processing based on clustering and morphological analysis. In *CRPITS '19: Proceedings of the First Asia-Pacific bioinformatics conference on Bioinformatics 2003* (2003), Australian Computer Society, Inc., pp. 111–118.

[10] Yang, J., Buckley, M., Dudoit, S., and Speed, T. Comparison of methods for image analysis on cDNA microarray data. *Journal of Computational and Graphical Statistics 11*, 1 (2002), 108–136.

[11] Yang, Y. H., Buckley, M. J., and Speed, T. P. Analysis of cDNA microarray images. *Briefings in Bioinformatics 2*, 4 (Dec 2001), 341–349.

Cloud Detection from Satellite Imagery: a Comparison of Expert-Generated and Automatically-Generated Decision Trees

Smadar Shiffman
Computational Sciences Division
NASA Ames Research Center / QSS
Moffett Field, CA 94035

Abstract

Automated cloud detection and tracking is an important step in assessing global climate change via remote sensing. Data products based on satellite imagery are available to the scientific community for studying trends in the Earth's atmosphere. The data products include cloud masks that assign cloud-cover classifications to pixels. Many cloud-mask algorithms have the form of decision trees. The decision trees employ sequential tests that scientists designed based on empirical astrophysics studies and simulations. Limitations of existing cloud masks restrict our ability to accurately track changes in cloud patterns over time. In this study we explored the potential benefits of automatically-learned decision trees for detecting clouds from remote-sensing images. The images were acquired using the Advanced Very High Resolution Radiometer (AVHRR) instrument on board the NOAA-14 weather satellite of the National Oceanic and Atmospheric Administration. We constructed three decision trees for a sample of 8km-daily AVHRR data from the year 2000 using a decision-tree learning procedure provided within MATLAB®, and compared the accuracy of the decision trees for sample test data to the accuracy of the cloud masks included in the AVHRR data products. We used ground observations collected by the National Aeronautics and Space Administration Clouds and the Earth's Radiant Energy Systems S'COOL project as the gold standard. For our sample data, the accuracy of automatically-learned decision trees was greater than the accuracy of the cloud masks. The difference in accuracy was statistically significant with $p < 0.001$.

1. Introduction

Understanding the role of clouds in the current climate is a prerequisite for predicting future climate change due to human activities [1]. The National Oceanic and Atmospheric Administration (NOAA) polar-orbiting satellites provide observations of the Earth's oceans, land, and atmosphere. Scientists use the observations to study long-term weather patterns and to forecast weather. The satellites carry a suite of instruments that measure parameters of the Earth's surface, atmosphere, and cloud cover. For example, the NOAA-14 satellite carries the Advanced Very High Resolution Radiometer (AVHRR) instrument. The data acquired by the satellites are packaged in a variety of data products of different spatial, temporal, and spectral resolutions. The data products are distributed via the NASA Goddard Earth Sciences Distributed Active Archive Center.

Cloud masks, which depict cloud coverage, have been included in data products from satellite radiometry since its early days. Scientists use cloud masks to identify surface and atmospheric data of compromised quality due to cloud interference and to describe clouds and their properties. The cloud masks are computed from measured reflectance and emission values using classification algorithms that scientists designed. The algorithms are based on experimentation with acquisition parameters and with simulated clear-sky and cloud characteristics for a variety of surface and atmospheric conditions, and on the analysis of ambiguous manifestations of different physical phenomena, for example, similar reflectance values for snow, ice and clouds. The algorithms employ sequential-threshold tests to arrive at decisions about the presence of clouds or about cloud composition [2-3]. The limitations of existing cloud masks [4] provided motivation for on-going research to develop improved cloud detection and characterization algorithms.

Cloud detection and characterization is a challenging task. Cloud-detection algorithms must disambiguate clouds and other entities that have characteristics similar to clouds. Entities whose appearance in satellite imagery may be similar to that of clouds differ from region to region. In the polar region, clouds and snow/ice are difficult to differentiate because all three entities are reflective in the visible wavelengths and demonstrate little contrast in the thermal infrared. Sun glitter due to spatially unresolved water bodies or recent rainfall may interfere with cloud detection in the tropics. Other entities that may appear similar to clouds are volcanic ash, desert dust, smoke, and terrain shadows.

Scientists have used a variety of machine-learning methods to process remote sensing data, for example, neural networks [5], Bayesian classification [6], kernel methods [7-10], genetic algorithms [11], classification trees [12] and regression trees [13]. The results of these approaches range from promising preliminary results to validated algorithms that are deployed in high-level remote-sensing data products [14]. From these methods, classification trees, which are automatically-learned decision trees, are the methods that resemble the sequential-threshold tests that are used in cloud masks the most.

In this study we combined elements from the sequential-threshold test approach that experts used to produce cloud masks, and from automatically-learned models that maximized classification accuracy for training data. The goal of our work was to determine whether automatically-learned decision trees that used the same variables as the CLAVR expert-generated cloud-mask algorithm performed better than the latter algorithm. We also explored the extent to which automatically-learned decision trees based on acquisition parameters and theoretical physical relationships among sensed data performed better than trees based on only sensed data.

The next section presents the cloud mask that is produced with an expert-generated decision tree, and discusses the limitation of the cloud mask. The section also lists the challenges in evaluating the results of cloud detection methods. Section 3 describes the methods we used in this study, Section 4 reports the results, and Section 5 includes a discussion of our findings. Section 6 presents our conclusions.

2. Background

The NOAA-14 AVHRR daily 8km global data product includes 12 scientific datasets (SDSs), each of which incorporates a measured parameter, flag, or computed parameter within a single plane. The SDSs are: normalized difference vegetation index, CLAVR cloud mask, quality control flag, scan angle, solar zenith angle, relative azimuth angle, surface reflectance in the visible wavelengths (channel 1), surface reflectance in the near-infrared wavelengths (channel 2), surface brightness temperature in the thermal infrared wavelengths (channels 3-5), and acquisition day and time [15].

The CLAVR algorithm includes four decision trees, one for each of daytime land scene, daytime ocean scene, nighttime land scene, and nighttime ocean scene. Each decision tree performs a series of threshold and uniformity tests on a 2x2 array of pixels, and classifies pixels as *clear*, *mixed*, or *cloudy*. The values used for each test are either retrieved channel values, or functions of retrieved values that incorporate acquisition parameters and estimates of emitted radiances [2]. Several tests were designed specifically to resolve ambiguities, for example, ambiguities due to reflectance greater than 44% in channel 1 or channel 2 for snow, ice, or sun glint. The thresholds used for the tests were derived empirically or via simulations of a variety of cloud-surface-daytime observation conditions.

The sequential decision process in CLAVR discriminates between clouds, first by their gross characteristics, and then by their subtle characteristics. The algorithm ensures that pixels that fail all the tests have a very small probability of having radiatively significant clouds. The sequential-test nature of CLAVR makes it similar to automatically-learned decision trees, but unlike the latter, the CLAVR algorithm is not based on an exhaustive analysis of the data space.

The CLAVR algorithm has several limitations. First, the algorithm assumes that there is a representative sample of clear pixels in each image, however, this assumption does not hold for broadly overcast scenes. Second, the algorithm does not work well for polar–winter scenes or nighttime scenes, when only the thermal channels are available. Third, the ability of the algorithm to differentiate between clouds and other entities that appear as clouds in AVHRR images is limited.

Evaluation of cloud masks is difficult because there is no gold standard to which to compare the mask. Researchers estimate the quality of cloud masks by comparing their agreement with masks produced by human analysts or by other algorithms. Stowe and colleagues [2] compared the results of CLAVR to cloud amount (fraction) estimates of a human–expert analyst. For this comparison, the CLAVR cloud mask categories were assigned cloud–cover percentages: clear-0%, mixed-50%, cloudy-100%. For small cloud amounts, CLAVR overestimated fractional amounts by approximately 10% compared to the analyst's interpretation, and for large cloud amounts, CLAVR underestimated the cloud amounts by approximately 10%. The evaluation showed larger errors for certain geographical locations and seasons. In a recent study, Thomas and Heidinger [16] reported that cloud amounts that resulted from improvements in CLAVR agreed with cloud amounts from established satellite–derived cloud climatologies.

In the next section we describe our work on learning decision trees from AVHRR data and comparing the cloud masks that they produce to ground observations performed by humans in multiple locations around the Earth.

3. Methods

We obtained ground observations from the NASA Langley Atmospheric Sciences Data Center CERES S'COOL Project [17]. High-school students from many geographical locations around the world recorded and reported the S'COOL observations using a well-defined protocol whose goal was to provide sufficient information for validation of measurements taken by the Clouds and the Earth's Radiant Energy System (CERES) instruments on NASA's Earth Observing System satellites. The observations were designed to coincide temporally with passage of the CERES above the point of observation. Among the recorded data were date and time of observation, longitude and latitude, cloud observations (categorical variables: cloud type, visual opacity; ordinal variables: cloud cover) at low, mid and high altitudes, and surface cover characteristics.

We selected all observations that were available for the year 2000. Then, we retrieved 8km daily AVHRR data that matched in acquisition date and in longitude and latitude. We excluded from this dataset all points for which the data quality flag indicated out-of-range values or processing errors (about 20% of the data points) and obtained 2869 data points. We used the S'COOL *cloud-present/no-cloud-present* observations for a given date and location as the gold standard for labeling training data and for comparing to classified test data. In addition, we compared the S'COOL observations to the CLAVR cloud masks for the retrieved data points. Although both CLAVR and the S'COOL project utilized an ordinal scale for characterization of cloud amount, the scales were not identical and mapping one scale to the other could be done in more than one way. Consequently, we mapped the CLAVR scale to a binary variable with values *clear* and *cloudy*, which we then could compare to the S'COOL binary variable with values *cloud-present/no-cloud-present*.

We performed three experiments with the AVHRR data that we selected. The experiments differed in the set of variables that constituted the input to the decision-tree learning procedure. Experiment I included variables that represented sensor data: observation identification, the radiances of channels 1 through 5, and the binary label that indicated the presence or absence of clouds obtained from the S'COOL data. Experiment II included the variables of Experiment I, as well as latitude, longitude, and acquisition parameters: scan angle, solar azimuth angle, relative azimuth angle, day of year and time of acquisition. Experiment III included the variables of Experiment II, as well as three additional function variables that are used within the CLAVR daytime-land algorithm [2]. Table 1 describes how the function variables related to sensed data and to acquisition parameters.

We randomly selected approximately 10% of the data points to form a dataset that would be used exclusively as a test set. We used bootstrapping [18] to learn and evaluate multiple decision trees from the remaining 2592 data points. For each trial, we randomly partitioned the data into a training set and a test set with a size ratio of 9:1. We learned a decision tree from the training set with the *treefit* procedure, which is an implementation of classification and regression trees [19] available within the MATLAB® statistics toolbox. We then classified the data in the corresponding test set as *clear* or *cloudy* using the decision tree. We compared the classification results to the S'COOL observations for matching date and location. To measure accuracy for each experiment we computed two mismatch rates. First, we computed the rate of mismatch between classification results of the automatically-learned decision trees and those of the S'COOL observations. Second, we computed the rate of mismatch between the CLAVR cloud masks and the S'COOL observations. We ran two-sided paired *t*-tests to determine if there were significant differences between rates of classification mismatch, for CLAVR and for each of the three decision trees, and for each pair of decision trees. Finally, we used the automatically-learned decision trees to classify the test set we had initially set aside, and we compared the rate of classification mismatch to that of CLAVR.

Table 1 Variables computed from sensed data and acquisition parameters

Test Name	Test Description	Variables	Function
Reflectance ratio cloud test (RRCT)	Examines the ratio of Channel 1 and Channel 2 reflectance	R_1 - Channel 1 reflectance R_2 - Channel 2 reflectance	$\text{RRC} = \frac{R_2}{R_1}$
Channel 3 albedo test (C3AT)	Extracts the reflectance component of the mixed Channel 3 signal	Spacecraft- dependent coefficients: a, b, c, d D_0 – Earth sun distance D – mean Earth-sun distance B – Planck blackbody radiance function v_o – Channel 3 central wave number T_i – observed equivalent blackbody temperature in channel i T_{3e}–estimated brightness temperature for Channel 3 due to emission only S_3–Channel 3 filtered solar irradiance at normal incidence and mean Earth-sun distance	$\text{C3A} = \frac{3.14159 \Delta R_3 \cdot 100}{\cos(Z_0)(D_o / D)^2}$ $\Delta R_3 = B(T_3) - B(T_{3e})$ $T_{3e} = -\left[(b/a)T_4 + (c/a)T_5 + d/a\right]$
Four-minus-five test (FMFT)	Examines the Channel 4 – Channel 5 brightness temperature difference	T_4 – Channel 4 brightness temperature T_5 – Channel 5 brightness temperature	$\text{FMF} = T_4 - T_5$

Note that each test compares the value computed by the corresponding function to a pre-defined threshold.

Table 2 Summary statistics for classification mismatch in the trials of each experiment

Exp.	Method	Mean (bootstrap)	Standard deviation (bootstrap)	Mean (independent)	Standard deviation (independent)
I	CLAVR	0.216	0.023	0.177	NA
I	Decision trees based on only channels	0.161	0.019	0.125	0.009
II	CLAVR	0.213	0.023	0.177	NA
II	Decision trees based on channels and acquisition parameters	0.152	0.019	0.131	0.017
III	CLAVR	0.210	0.021	0.177	NA
III	Decision trees based on channels, acquisition parameters, and functions	0.151	0.022	0.105	0.013

There are three different mean (bootstrap) values for CLAVR because the random partitions of the data into training and test sets were different in each experiment. The independent test-set evaluation with CLAVR produced a single classification mismatch rate, and therefore the mean value is constant for the three experiments, and the standard deviation is zero.

4. Results

Table 2 lists the mean and standard deviation trial classification mismatch for each experiment. Note that the bootstrapping training sets were not independent, and the test sets were not independent as well. However, the test set that was set aside initially was independent of all other sets. The two-sided paired *t*-test showed that the differences in classification-mismatch rates between CLAVR and each of the automatically-learned decision trees was significant in all three experiments with $p < 0.001$. The differences in classification-mismatch rates among the automatically-learned decision trees were not statistically significant. Figure 1 shows an example decision tree learned in Experiment I.

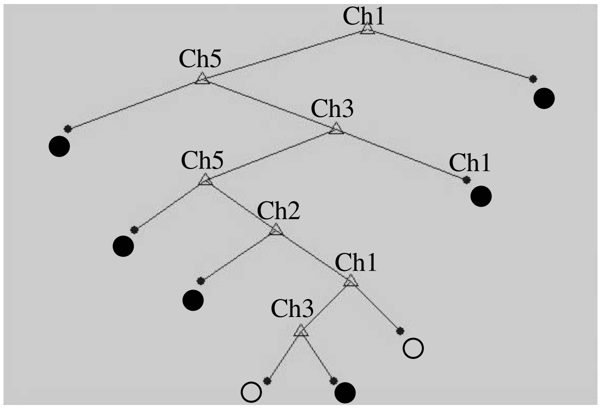


Figure 1 An example decision tree used in the first experiment. The labels for non-terminal nodes indicate the channels used in each decision point (we omitted the thresholds used in each node for brevity). The terminal nodes show the assigned label: A filled circle indicates *cloudy* and a clear circle indicates *clear*.

5. Discussion

The three experiments that we performed showed that automatically-learned decision trees classified 8km daily AVHRR data for the year 2000 more accurately than CLAVR. The three types of decision trees—trees based on only sensor data, trees based on sensor data and acquisition parameters, and trees based on sensor data, acquisition parameters, and functions of the sensor data and the acquisition parameters—did not show significant differences in classification accuracy. Thus the sensor data alone were sufficient to obtain an improvement over CLAVR. Note that the prevalence of cloudy pixels in the sample dataset was about 80%. CLAVR did as well as a classifier that always assigned the label *cloudy*, while all decision trees performed better than the latter classifier by about 25%.

The data we used for our study was a sample of convenience. The location and dates of S'COOL ground observation dictated which sensor data were included in the experiment. The validity and reliability of the S'COOL observations are currently under study [17]. Because the S'COOL observations did not necessarily coincide temporally with the exact time in which AVHRR passed above the observation point, cloud patterns and illumination may have changed between the observation time and the satellite-passage time. We assumed that any differences that resulted from the gap between AVHRR acquisition time and S'COOL observation time affected CLAVR and automatically-learned decision trees equally, and therefore we disregarded these potential differences. We used the S'COOL observations as a best-estimate of the gold standard because there is no gold standard data for the presence or absence of clouds. We classified the AVHRR data into only two classes, *cloudy* and *clear*. Finer cloud categories are available in both the CLAVR mask (*clear*, *mixed*, and *cloudy*) and the S'COOL data (*clear*, *partly cloudy*, *mostly cloudy*, and *overcast*). We did not use the finer categories because mapping the CLAVR categories to the S'COOL categories was not straightforward. Note that, in effect, the automatically-learned decision trees were trained to predict S'COOL observations from AVHRR data. Thus, our ability to conclude the true presence or absence of clouds based on the results of automatically-learned decision-trees is limited by the accuracy of the S'COOL observations.

The automatically-learned decision trees are similar to the CLAVR algorithm in that they use individual channel variables multiple times, with a different threshold each time, to refine the classification. However, unlike the CLAVR algorithm, the automatically-learned decision trees do not examine the uniformity of a 2x2 spatial neighborhood to reduce the uncertainty and to reach a final decision about the class of a given data point. The automatically-learned decision trees were powerful in that, for our sample data, they were able to learn a compact model (small number of channel variables) that produced better predictions than CLAVR. If the S'COOL data were available to the designers of the CLAVR algorithm, they might have been able to tune CLAVR to generate better predictions of S'COOL observations.

While the CLAVR design process did include careful study and fine tuning of test thresholds, the process explored only part of the solution space. However, the decision-tree learning process performed an extensive search to find a sequential-threshold solution that optimized the prediction accuracy for the training set. We could explain the difference in performance between the two methods in that the automated learning process could have captured explicitly structure and relationships between variables that were inherent in the problem domain and that might not have been perceived as relevant by the CLAVR experts.

The experiments that we performed learned multiple decision trees, one for each sample training set. Hypothetically, if we were to deploy a learned decision tree for cloud detection within the context of a remote sensing data product, it would not be obvious which of the multiple decision trees to deploy. Constructing an ensemble of decision-tree models [20] would allow us to obtain a robust classifier that could potentially perform better than any one decision tree.

6. Conclusion

We demonstrated that automatically-learned decision trees predicted the presence or absence of clouds more accurately and efficiently than CLAVR for our selected dataset. To further validate our results we will replicate this study using gold standard data obtained from cloud-observations made by scientists at surface weather stations. A more comprehensive replicate study, for multiple-year data with a range of spatial and temporal resolutions, acquired under different surface conditions (ocean, land, daytime and nighttime settings) and with potential ambiguity due to the presence of smoke or water-induced glitter, would allow us to generalize our results. Two possible extensions of this work are to use automatically-learned decision trees to classify clouds into multiple could types, and to use regression trees to predict the amount of cloud cover and visual opacities.

Acknowledgements

The gold-standard data for this study were obtained from the NASA Langley Research Center Atmospheric Sciences Data Center as part of the CERES S'COOL Project. We thank Ashok Srivastava, Mark Schwabacher, Nikunj Oza, Dawn McIntosh, and Marianne Mosher for useful discussions and comments on earlier version of this manuscript. This work was funded by the Intelligent Systems / Intelligent Data Understanding program for Virtual Sensors.

References

[1] Wielicki, B.A., Harrison, E.F., Cess, R.D., King, M.D., Randall, D.A. Mission to planet earth: role of clouds and radiation in climate, *Bulletin of the American Meteorological Societ*y, Vol. 76, No. 11, pp. 2125-2154, 1995.

[2] Stowe, L.L., Davis, P.A. Davis, and McClain, E.P. Scientific basis and initial evaluation of the CLAVR-1 global clear/cloud classification algorithm for the advanced very high resolution radiometer. *Journal of Atmospheric and Oceanic Technology,* Vol. 16(6), pp. 656-681, 1999.

[3] Trepte, Q.Z., Minnis, P., and Arduini, R.F. Daytime and nighttime polar cloud and snow identification using MODIS. In Huang, H-L., Lu, D., Sasano, Y. *Proc. SPIE 3rd International Asia-Pacific Environmental Remote Sensing Symposium 2002: Remote Sensing of the Atmosphere, Ocean, Environment, and Space*, Hangzhou, China, October 23-27, pp. 449-459, 2003.

[4] Stroeve, J. Assessment of greenland albedo variability from the avhrr polar pathfinder data set, *Journal of Geophysical Research*, Vol. 33, pp. 989–1034, 2002.

[5] Promcharoen, S., Rangsanseri, Y., Suwit Ongsomwang, S., Jaruppat, J. Supervised classification of multispectral satellite images using fuzzy logic and neural network. In *Proceeding of the 20th Asian Conference on Remote Sensing*, November 22-25, Hong Kong, China, 1999.

[6] Murtagh, F., Barreto, D. Marcello, J. Decision boundaries using Bayes factors: the case of cloud masks. *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 14, pp. 2952-2958, 2003.

[7] Lee, Y., Wahba, G., Ackerman, S.A. Cloud classification of satellite radiance data by multicategory support vector machines. *Journal of Atmospheric and Oceanic Technology*, Vol. 21, No. 2, pp. 159-169, 2004.

[8] Srivastava, A.N. Mixture density Mercer kernels: a method to learn kernels directly from data. In M.W. Berry, U. Dayal, C. Kamath, and D. Skillicorn. (Eds.) *Proceedings of the Fourth SIAM International Conference on Data Mining*. Lake Buena Vista, Florida, April 22-24, 2004.

[9] Srivastava, A.N., Oza, N.C., and Stroeve, J. Virtual sensors: using data mining techniques to efficiently esimate remote sensing spectra. *IEEE Transactions on Geosciences and Remote Sensing, Advances Special Issue*, 2005. Accepted.

[10] Garay, M.J., Mazzoni, D., Davies, R., and Diner, D.J., The application of support vector machines to analysis of global satellite data sets from MISR. In *Proceedings of the 4th AMS Conference on Artificial-Intelligence Applications to Environmental Science*, San Diego, Jan 9-13, 2005.

[11] Brumby S.P., Hirsch K.L., Davis A.B., Harvey N.R., and Rohde C.A. Genetic refinement of cloud-masking algorithms for the multi-spectral thermal imager (MTI). In *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, Sydney, Australia, July 9-13, pp. 1152-1154, 2001.

[12] Hansen, M., Dubayah, R., and DeFries, R. Classification trees: an alternative to traditional land cover classifiers. *International Journal of Remote Sensing,* Vol. 17, pp. 1075-1081, 1996.

[13] Schwabacher, M., Langley, P. Discovering communicable scientific knowledge from spatio-temporal data. In *Proceedings of the Eighteenth International Conference on Machine Learning.* pp. 489-496. Williamstown, MA: Morgan Kaufmann, 2001.

[14] Zhan, X., Sohlberg, R.A., Townshend, J.R.G, DiMiceli, C., Carroll M.L., Eastman, J.C., Hansen, M.C. , DeFries, R.S. Detection of land cover changes using MODIS 250m data. *Remote Sensing of Environment*, Vol. 83, No. 1-2, pp. 336–350, 2002.

[15] Agbu, P.A. and James, M.E. *The NOAA/NASA Pathfinder AVHRR Land Data Set User's Manual.* Goddard Distributed Active Archive Center, NASA, Goddard Space Flight Center, Greenbelt, MD, 1994.

[16] Thomas, S., Heidinger, A.K. and Pavolonis, M.J. Comparison of NOAA's operational AVHRR derived cloud amount to other satellite derived cloud climatologies. *Journal of Climat*e, Vol. 17, No. 24, pp. 4805-4822, 2004.

[17] Chambers, L.H., Costulis, P.K., Young, D., and Rogerson T.M.. Students as ground observers for satellite cloud retrieval validation. In *Proceedings of the 13th Conference on Satellite Meteorology and Oceanography*, Sep 20-23, 2004, Norfolk, VA.

[18] Mooney, C.Z., and Duval, R.D. *Bootstrapping: A Nonparametric Approach to Statistical Inference*. Newbury Park, CA: Sage Publications, 1993.

[19] Brieman, L., Friedman, J., Stone, C.J., and Olshen, R.A. Classification and regression trees. Chapman & Hall, CRC Press, Boca Raton, FL, 1984.

[20] Dietterich, T.G., Ensemble methods in machine learning. In J. Kittler and F. Roli, (Eds.), *First International Workshop on Multiple Classifier Systems*, pp. 1–15. Springer-Verlag, 2000.

A MISR cloud-type classifier using reduced Support Vector Machines

Dominic Mazzoni[1], Ákos Horváth[1], Michael J. Garay[2], Benyang Tang[1], and Roger Davies[1]

1. Jet Propulsion Laboratory, California Institute of Technology,
Pasadena, CA 91109, USA
`Dominic.Mazzoni@jpl.nasa.gov`

2. University of California, Los Angeles
Los Angeles, CA 90095, USA
`garay@atmos.ucla.edu`

Abstract. We are developing a pixel-level cloud-type classifier for the Multi-angle Imaging SpectroRadiometer (MISR), an instrument used to study clouds and aerosols from NASA's Terra satellite. To augment MISR's existing high-level products (including cloud masks, cloud heights, and aerosol optical depth retrievals), our cloud-type classifier labels each 1.1-km pixel as clear, or as belonging to one of several types of cloud. In the past, similar classifiers have been developed for other remote-sensing instruments using various machine learning techniques, such as artificial neural networks. However, support vector machines (SVMs) are not typically used, in part because the computational cost of evaluating new examples with an SVM can be much higher. Our novel approach to achieving high classification accuracy within the computational requirements of the operational MISR processing system involves training a very large multi-class SVM using thousands of training points and then applying cutting-edge reduced-set techniques to yield a computationally manageable number of support vectors. The resulting product will help provide new insights for those constructing cloud climatologies, modeling radiative transfer through clouds, and studying the effects of clouds on climate, in addition to demonstrating the effectiveness of using SVMs in a production science setting.

1 Scientific Motivation

Because different cloud types are formed by different mechanisms, cloud type is often indicative of underlying atmospheric processes. In that regard, in satellite data analysis it is useful to separate stratiform (layered) clouds from cumuliform (puffy) clouds, at the very minimum. This is because the radiative and hydrological impacts of these two cloud types are very different. Stratiform clouds are thin and relatively dry clouds that cover large areas. They have a profound effect on the radiative properties of the earth-atmosphere system due to their ability to reflect large amounts of solar radiation back to space. However, stratiform clouds contain only a very small portion of the total atmospheric liquid water. Cumuliform clouds behave in the opposite manner. These clouds cover small areas and have only a small radiative impact, but contain most of the atmospheric liquid water. Additionally, these two cloud types are indicative of the direction of atmospheric heat transport: stratiform clouds tend to transport heat horizontally, while cumuliform clouds represent heat transport primarily in the vertical.

A second motivation for classifying satellite observations of clouds into different cloud types is that this allows satellite observations to be related to surface-based observations. Surface weather observers are trained to group clouds into 10 standard types and extensive global compliations exist of these observations. Comparison between satellite and surface observations is facilitated by grouping satellite measurements into similar categories. The International Satellite Cloud Climatology Project (ISCCP), for example, attempts to do this by associating exactly one cloud type with particular ranges of cloud-top height and cloud optical thickness [1]. However, comparisons of ISCCP and surface climatologies (e.g., Hahn et al. [2]) have found that this simple approach does not quite work properly and ISCPP can only reliably distinguish a limited set of combined cloud classes. Clearly, it would be desirable to do better.

An increasingly important motivation for cloud classification in satellite imagery is climate monitoring. Global changes in the amounts of different cloud types is a potential signal of climate change. Within the operational forecast community as well there is interest in methods of automatic satellite cloud classification. Although weather forecasters typically rely on imagery from geostationary satellites, such as the Geostationary Operational Environmental Satellites (GOES), techniques for automatic cloud classification developed for other satellite instruments, such as the Multi-angle Imaging SpectroRadiometer (MISR) onboard NASA's Terra satellite, could potentially lead to new insights that could be incorporated into an automatic cloud classifier for GOES imagery.

2 Related Work

Perhaps the most visible example of automatic cloud-type classification using machine learning is Bankert's real-time classifier for GOES images available on the web [3]. While the classification algorithm it uses is simple (1-nearest-neighbor), the features were selected carefully using a backward sequential selection (BSS) algorithm, and the system boasts an impressively large collection of training data (10 cloud types classified in over 5,000 GOES scenes). More details about the approach are described in a Tag, Bankert et al. paper on Advanced Very High Resolution Radiometer (AVHRR) cloud-type classification [4]. Overall their approach leads to fast and accurate classification. However, some shortcomings are apparent, including limited detection of thin cirrus clouds and small cumulus clouds, as well as multilayer systems of cirrus over low clouds being misclassified as mid-level clouds. The latter problem, in particular, occurs when low and high cloud infrared temperatures are averaged to obtain the associated cloud height, a problem that would be alleviated if cloud heights could be determined by another method.

Dozens of other papers exist on automatic satellite cloud-type classification using learning algorithms. Some pioneering work was done by Welch et al. [5], comparing the use of discriminant analysis and two types of neural networks to classify pixels in AVHRR images as one of a number of classes, including five cloud types. Bankert et al. [6] [7] [8] compared neural networks to decision trees and a 1-nearest-neighbor classifier in GOES data. Tian et al. [9] used probabalistic neural networks to classify 10 cloud types in GOES images using both spatial and temporal features. Saitwal et al. [10] extended Tian et al.'s work to nighttime classification. Baum et al. [11] used fuzzy logic to detect multilayer systems in AVHRR scenes based on examples getting classified into more than one of eight trained cloud types. Azimi-Sadjadi and Zekavat [12] used a hierarchical arrangement of support vector machines to classify six different cloud types in infrared GOES-8 imagery, while Lee et al. [13] investigated using a multi-class support vector machine to distinguish between ice and water clouds in MODIS images. Li et al. [14] used the maximum likelihood technique to improve on the basic cloud classification provided in the MODIS standard product.

3 Background

We used Support Vector Machines (SVMs) to construct a cloud-type classifier for image data from the Multi-angle Imaging SpectroRadiometer (MISR) satellite instrument. This section provides a brief overview of SVMs and the characteristics of the MISR instrument.

3.1 Support Vector Machines

SVMs [15] are a popular technique for supervised classification. Training a binary SVM is a quadratic optimization procedure that finds an optimal hyperplane separating positive and negative example vectors, balancing accuracy with generalization performance by maximizing the margin. The margin is simply the distance from the hyperplane to the nearest positive and negative labeled feature vectors. However, since most problems are not linearly separable, SVMs typically use a kernel function that implicitly projects two example vectors in input space (X_i, X_j) into (possibly infinite) feature space vectors $(\Phi(X_i), \Phi(X_j))$ and returns their dot product in that feature space. Popular kernels (with model selection parameters a, p, σ) include:

$$\begin{array}{ll} \text{linear:} & K(u,v) = u \cdot v, \\ \text{polynomial:} & K(u,v) = (u \cdot v + a)^p, \\ \text{RBF:} & K(u,v) = \exp(-\frac{1}{2\sigma^2}\ ||u-v||^2) \end{array}$$

It is also common to use normalized kernels, which makes it more practical to work with high-degree polynomials:

$$K_{\text{norm}}(u,v) := K(u,v)K(u,u)^{-\frac{1}{2}}K(v,v)^{-\frac{1}{2}}$$

The hyperplane resulting from SVM training is defined implicitly by a vector of weights α_i on the original training examples. Many of these α_i will be zero and the associated example vectors can be discarded. The example vectors whose corresponding α_i is nonzero – called *support vectors* – are needed in order to determine the classification of new points. Note that when a linear kernel is used, it is possible to compute the hyperplane normal vector by summing the α-weighted support vectors. For any nonlinear kernel, though, the hyperplane normal vector exists in feature space and may not have a pre-image in input space. As a result, classifying a new example using a SVM can be expensive, requiring one dot-product calculation per support vector. Because more difficult classification problems tend to require more support vectors, SVMs have developed a reputation for being powerful but inefficient.

SVM reduced-set methods [16] use various techniques to decrease the number of support vectors, either by eliminating those support vectors that contribute the least (and re-weighting the remaining ones) or by explicitly solving for the pre-image of the hyperplane normal vector. While previous results have shown impressive reductions in the number of support vectors with minimal loss of generalization performance on some problems, many other problems proved to be irreducible, and the reduction algorithms were typically very slow. However, recent breakthroughs (described in greater detail below) have resulted in even more sophisticated reduced-set techniques that overcome many of these limitations.

Although there have been methods developed for "true" multi-class SVMs (e.g., [17]), it is usually more practical to perform multi-class classification by training several binary SVMs [12] [18]. Common methods for solving multi-class problems using binary SVMs include one-vs-one, one-vs-all, and directed acyclic graph (DAG).

3.2 MISR

The Multi-angle Imaging Spectroradiometer (MISR) is one of five instruments aboard NASA's Terra satellite, which follows LandSat 7 in polar orbit around the Earth at an altitude of 705 km with an equatorial crossing time of 10:30 am (local time) for the descending portion of the orbit. MISR has nine pushbroom cameras pointed in different directions along the orbital path, ranging from 70° forward to 70° aftward. Each camera views four spectral bands (blue, green, red, and near-infrared) with a resolution of 1.1 km × 1.1 km per pixel (plus limited coverage at 275 m × 275 m). The images are projected to a common grid and coregistered during automatic ground processing, resulting in nine views of each scene with a swath width of about 350 km.

There are at least three ways that MISR's multiple angles yield cloud information not available via conventional satellite means. First, thin clouds and aerosols are more opaque at oblique angles because the photon path length is longer, making these atmospheric constituents more apparent against the background surface. Second, objects in MISR imagery can be identified by their angular signature. Because aerosols and clouds scatter radiation differently into different directions at different wavelengths, this knowledge can be exploited to help characterize clouds and aerosols. Third, because the cameras are registered to the surface ellipsoid, objects above the surface appear displaced due to the parallax effect (see Figure 1). Operationally, an automatic pattern-matching algorithm is used in the instrument software to determine the disparity of each pixel and infer the height using triangulation. This is complicated by the fact that there is a seven minute delay between the imaging times of the first and last cameras to view each scene and approximately a one minute delay between images taken by each camera, during which time clouds may have moved. However, the cloud-top height (± 500 m at a spatial resolution of 1.1 km) and the height-resolved mesoscale wind (± 4 m/s at a spatial resolution of 70.4 km) can be retrieved operationally by various stereoscopic combinations of views [19] [20]. The stereoscopically-derived height provides an independent check against the cloud-top height measurements made by other instruments using measurements of emitted radiation and assumptions about the thermal structure of the atmosphere.

While several methods have been developed to obtain scientifically useful measurements from multi-angle satellite data, the full parameter space of information remains largely unexplored. We have found that machine learning techniques allow us to explore this parameter space much more quickly.

4 Previous Work

This work builds on our previous successes developing pixel classifiers for MISR [21]. We created a binary cloud mask (distinguishing cloudy pixels from clear pixels) using a total of 156 features as input.

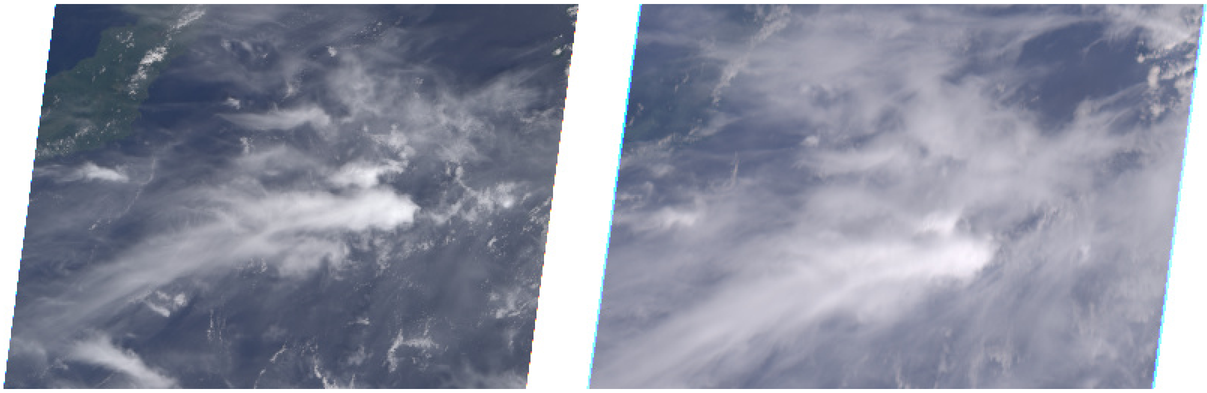


Fig. 1. On the left, MISR's view of multilayer clouds over the Molucca Sea, with the northeastern tip of the Indonesian island of Sulawesi in the upper-left of the image, taken on December 1, 2004. On the right, MISR's 70-degree forward view of the same scene. Note both the increased opacity of the cirrus clouds and the vertical displacement of the clouds due to their height above the surface. Path 111, Orbit 26354, Blocks 89-90, courtesy Langley Atmospheric Sciences Data Center.

The features were derived from raw MISR radiances from all four spectral bands and the three most nadir-pointing cameras, as well as several neighboring pixels for context. We trained two separate SVMs: one specializing in clouds over water and one over land (the MISR standard product identifies whether each pixel is over water or over land). We used two of MISR's existing cloud masks to provide training labels for these two SVMs, but only over surfaces where we knew the existing cloud masks tended to be accurate. The resulting SVMs we trained over water and land had 656 and 2456 support vectors, respectively. To validate the performance, we independently labeled 3,500 pixels randomly distributed throughout the globe, finding that the existing cloud masks each had error rates of approximately 12% relative to these expert labels, while our SVM cloud mask had an error rate of approximately 6%.

Subsequently, we developed improved reduced-set techniques that successfully reduced both SVMs used in the cloud mask to 20 support vectors each. The accuracy of the land SVM remained unchanged, while the accuracy of the water SVM decreased by only 0.3%. The resulting classifier requires just slightly more than 156×20 multiply-add instructions per pixel, which we determined was fast enough to run as part of the standard MISR data processing. We have completed a test of processing the SVM cloud mask at the NASA Langley Atmospheric Sciences Data Center and are working on integrating it and making it available as a standard MISR product later this year.

We also developed two other binary classifiers for MISR using the same approach. Thin cirrus clouds can have potentially large radiative effects on the atmospheric and surface energy budgets and they present an impediment to the operational retrieval of clear sky atmospheric and surface properties. However, thin cirrus clouds are notoriously difficult to detect using standard satellite remote sensing techniques. Our SVM cirrus cloud detector was trained using expert labels with essentially the same inputs as the cloud mask described above. However, instead of using the three most nadir pointing MISR cameras, we used the three most forward cameras in the northern hemisphere and three most aftward cameras in the southern hemisphere. This approach took advantage of the unique capabilities of MISR relative to other instruments by utilizing the increased path length of photons being scattered in the forward direction at these oblique viewing angles. Finally, a SVM smoke detector was developed to distinguish smoke and some related aerosols from cloudy and clear pixels in MISR imagery. Because the detection, as opposed to the precise location, of the smoke was of primary interest, we experimented with using data from five of MISR's nine cameras as input. This resulted in improved detection of smoky regions due to the increased photon path length for the oblique cameras used, as well as the characteristic angular signature of smoke, which was made more apparent through the use of a larger number of angles. These two classifiers have been validated, but not yet to the same degree of precision as the cloud mask. We hope to finish the validation effort and make both these classifiers available as MISR data products later in the year, as well.

5 Methodology

Our previous three MISR classifiers were binary (e.g. cloudy vs. clear). As our goal in this new research was to develop a cloud-type classifier, we needed to deal with many new issues associated with a multi-class learning problem, such as how to handle examples that could conceivably belong to more than one class. After choosing the classes to label, we began by creating expert labelings of 30 scenes, each one approximately a quarter of a MISR swath, about $400 \times 5,000$ pixels. Our strategy was not to label every pixel, nor to choose individual scattered pixels; instead we used a graphical interface to rapidly "paint" labels over the top of the image wherever we were reasonably confident about the classification, avoiding cloud edges and multilayer systems. Using this technique we labeled over 3 million pixels across the 30 scenes. With practice, we found that MISR is particularly amenable to expert labeling by visual inspection, because we could rapidly flip between the different camera views of one scene and see its unambiguous three-dimensional structure.

We originally considered 12 classes, made up of 4 non-cloud classes and 8 cloud types, but later combined some of the classes that were difficult to distinguish, resulting in the choice of the following eight classes:

Abbrev.	*Class*
`Ocean`	Clear Ocean
`Land`	Clear Land
`Ice`	Ice or Snow
`m Sc`	Marine Stratus or Stratocumulus
`sm Cu`	Small Cumulus
`Cb`	Cumulus Congestus or Cumulonimbus
`Ci`	Cirrus
`St`	Other Stratiform

Examples from every class showed up in at least three distinct scenes.

To construct the feature vectors for classification, we started with the feature vector we used successfully for cloud detection, which will now be described in greater detail. One of the principal ideas behind our feature vectors is deceptively simple: in order to classify a single pixel, we use all the values from a 5×5 (or larger) region of pixels surrounding the central pixel as features. One source of inspiration for this idea was the observation that SVMs were able to successfully classify images of handwritten digits from the MNIST database with a feature vector consisting simply of all of the 27×27 grayscale values making up the image [22].

Using these feature vectors, we trained a multi-class SVM using the one-vs-all method. Of the 30 labeled scenes available, we arbitrarily chose 24 for training and 6 as holdouts for validation.

To choose the SVM hyperparameters (the kernel function and regularization parameter C), we chose 2,000 training examples (250 from each class) and 2,000 test examples. We then exhaustively searched a space of several dozen kernel functions and values of C from $0.1 \ldots 3000$, training a multi-class SVM with each set of parameters. We chose the kernel and C combination that had roughly the lowest test error. However, all other things being equal, we favored parameters that resulted in fewer support vectors, based on our general observation that the same accuracy with fewer support vectors is more likely to generalize. The best choice turned out to be a normalized polynomial kernel with $p = 17$ and $a = 1$, and $C = 50$.

After selecting the hyperparameters, we trained a larger SVM using 8,000 training examples (1,000 from each class) using the one-vs-all method. (We also tried one-vs-one and got similar results, but one-vs-all was preferable for reducing, as will be seen later.) On a holdout set of 8,000 examples, the overall accuracy was 78.6%, with the following confusion matrix:

	Ocean	Land	Ice	m Sc	sm Cu	Cb	Ci	St
Ocean	88.2%	0.0%	0.0%	0.6%	10.5%	0.0%	0.7%	0.0%
Land	0.0%	99.7%	0.0%	0.0%	0.1%	0.0%	0.2%	0.0%
Ice	0.0%	0.0%	58.5%	0.2%	0.8%	21.2%	13.1%	6.2%
m Sc	0.5%	0.0%	0.1%	66.7%	10.1%	13.6%	3.7%	5.3%
sm Cu	18.1%	2.9%	0.1%	12.5%	61.9%	0.0%	4.5%	0.0%
Cb	0.0%	0.0%	4.0%	7.0%	0.1%	87.1%	1.5%	0.3%
Ci	1.2%	0.3%	0.2%	5.4%	7.2%	2.9%	82.8%	0.0%
St	0.0%	0.0%	0.0%	4.4%	0.0%	9.9%	0.7%	85.0%

From the confusion matrix, we gathered that the most difficult classes to separate were ice, marine stratocumulus and small cumulus. Interestingly, we found that ice was often being misclassified as cumulonimbus, perhaps because they are both highly reflective surfaces. However, it would be easy to distinguish between these two classes with additional height information available from MISR.

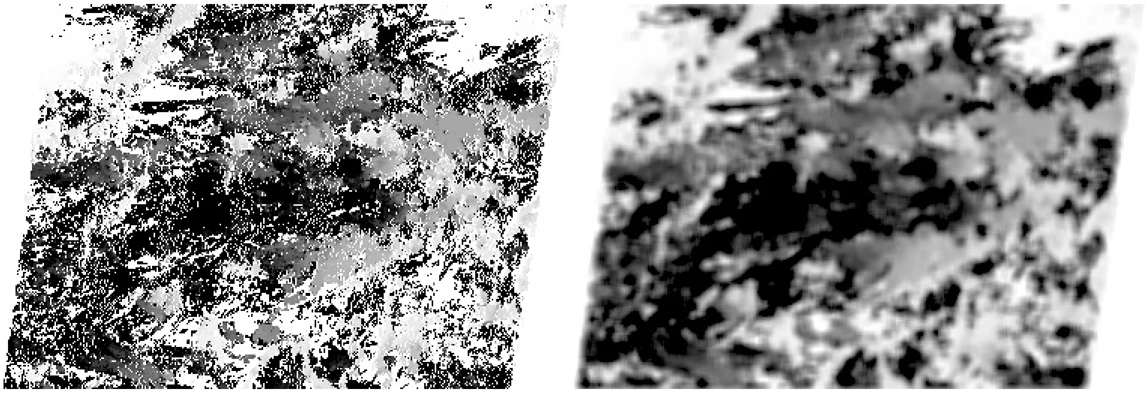


Fig. 2. On the left is the image of the automatically generated stereo height product for the MISR image seen in Figure 1. Darker-colored pixels are higher above the surface, up to a maximum of about 12 km in this particular scene. The noise is mainly due to blunders from the stereo matching algorithm. On the right is the smoothed stereo image used to provide an additional feature for our SVM.

The misclassification of small cumulus as ocean (and vice versa) is not unexpected. Small cumulus cloud fields are often broken, which allows the ocean surface to be seen between clouds. This will create particular difficulties at the edges of cloud fields. The vertical development of these clouds also introduces a parallax effect that can cause the classifier to incorrectly classify pixels as cloudy because clouds are seen in one camera and not another.

5.1 Incorporating stereo height

Initially we were concerned about incorporating height information because MISR's stereo-derived height product, while extremely accurate overall, tends to be noisy at the pixel level. Therefore, we developed an appropriate smoothing algorithm for this product to allow its use as an additional input for our feature vectors. An image of the stereo height product before and after smoothing can be seen in Figure 2.

Adding the smoothed stereo height to our feature vector resulted in significant improvement in the classifier performance as shown in the confusion matrix below. Overall, after incorporating stereo height, the accuracy improved from 78.6% to 85.5%:

	Ocean	Land	Ice	m Sc	sm Cu	Cb	Ci	St
Ocean	87.9%	0.0%	0.0%	0.6%	8.5%	0.0%	3.0%	0.0%
Land	0.0%	99.9%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%
Ice	0.0%	0.0%	87.2%	1.0%	1.5%	1.7%	2.5%	6.1%
m Sc	0.4%	0.0%	0.3%	71.0%	9.8%	7.7%	4.1%	6.7%
sm Cu	14.3%	4.3%	0.0%	12.8%	68.1%	0.1%	0.4%	0.0%
Cb	0.0%	0.0%	0.1%	6.0%	0.2%	92.1%	1.4%	0.2%
Ci	0.4%	0.4%	0.0%	3.0%	1.7%	2.9%	91.6%	0.0%
St	0.0%	0.0%	0.0%	1.6%	0.0%	9.4%	1.2%	87.8%

The confusion matrix shows that the most challenging classes all improved their accuracies significantly.

5.2 Smoothing the results and applying the cloud mask

The resulting multi-class SVM classified each pixel as one of eight different classes. Based on visual inspection of the results, in many regions the classification appeared to be quite good. However, we found that the resulting classification images were quite noisy. As was done in [4] and several other cloud-type classifiers, we attempted to smooth the resulting image and avoid noise. The results of our smoothed classification can be seen in Figure 3.

Rather than just using the class of each pixel in the smoothing, we found it was advantageous to use the relative weights of each binary SVM in the smoothing process, taking advantage of the fact that we were using a one-vs-all multi-class SVM. In the one-vs-all paradigm, one binary SVM for each class is designed to determine whether the example is in that class, or in one of the other classes. The output of the binary SVM is a scalar, indicating the distance between that example and the decision hyperplane, with a larger

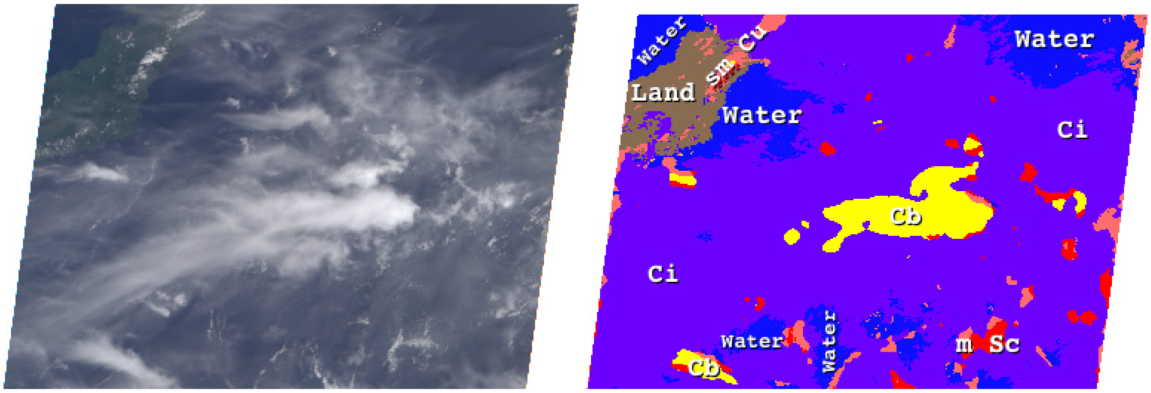


Fig. 3. On the left is the same MISR image seen in Figure 1. On the right, the result of our final, smoothed cloud-type classification algorithm, which has detected regions of cumulonimbus (Cb), cirrus (Ci), some small cumulus clouds (sm Cu), some small marine stratocumulus (m Sc), water and land.

value indicating a farther distance from the hyperplane and the sign indicating whether it falls on the positive or negative side. Class membership is then decided by taking the argmax of all of the binary SVMs.

So for every pixel we have a vector of eight binary SVM outputs, one for each of our eight classes. In the simplest case, one of these outputs will be positive and the other seven will be negative, indicating that the pixel in question clearly belongs to one class. However, for ambiguous pixels, or those on the border between regions, several classes may have positive values, in which case the largest one is chosen as the most likely class. In order to smooth the classifier, then, we averaged each vector with the vectors of several neighboring pixels, with a neighbor contributing less depending on the square of its distance from the pixel in question. After this averaging, the argmax of each vector was taken to determine the class.

To better understand the consequences of this type of smoothing, consider a single, isolated pixel classified as class 1, when all of its neighbors were classified as class 2. If the isolated pixel was very confidently class 1 (the class 1 binary SVM output was very large, and the other values were very negative), the averaging would not change anything. However, if the isolated pixel was right on the border, perhaps with classes 1 and 2 having almost equal weight, then the averaging would change the pixel's classification to class 2. We felt that this strategy did the best job of encouraging regions to be more homogeneous while allowing small heterogeneous regions when they were detected with above average confidence. One could imagine making the degree of smoothing a tunable parameter.

Finally, in generating the final cloud-type classification product, we made use of our existing SVM cloud mask and the existing MISR land/water classification to refine our results. The idea is that when training our cloud-type classifier, we were focusing on how best to distinguish between different types of clouds, and less on how to determine whether a pixel was cloudy or clear. Especially after our smoothing, we found that far more pixels were classified as some type of cloud than would normally be considered cloudy pixels. Therefore, as an additional pass, we replaced each pixel which our SVM cloud mask marked as "clear," with either the clear land or clear water class, as appropriate.

5.3 Reducing the multi-class SVM

Several higher-level products are generated from the raw radiance data and distributed along with MISR data. These include several cloud masks, the stereo-derived height products, aerosol optical depths, and top-of-atmosphere albedos. Our goal was to create a classifier that was computationally fast enough that we could integrate it into the data processing system that automatically generates MISR products at the Langley Atmospheric Sciences Data Center. While it was difficult to determine the exact amount of processing time potentially available, very roughly we determined that we could run a single binary SVM that gathered approximately 150 features and used in the vicinity of 200-300 support vectors. Unfortunately, the multi-class SVM we had trained to do cloud-type classification had 3,340 support vectors (total, including all eight binary SVMs in the one-vs-all framework), at least one order of magnitude too many.

Some of our previous research includes methods to classify new examples using SVMs more efficiently [23] by reordering the support vectors on the fly, skipping most of the support vectors for "easy" examples, and only using the full SVM for the most difficult (ambiguous) examples. While we have seen speedups as high as 12× with this technique, actual speedups on this particular problem were less, and any efficient implementation of this algorithm is enormously more complicated than the normally straightforward SVM evaluation formula, making it a poor candidate for integrating into a production software system.

Instead, we investigated the use of reduced-set methods [16], which attempt to approximate the hyperplane normal vector using a small number of vectors instead of all of the support vectors. While we found the theory to be quite solid, the pre-image algorithm in [16], which maps a feature space vector back to the input space, often got stuck in local minima, making the reduction process very frustrating. Therefore, we developed our own pre-image algorithm, using Differential Evolution [24] to find a rough pre-image, then applying gradient descent to refine it. Another improvement we made is that after the reduced-set vectors have been computed we readjust the weights and bias through a modified SVM training algorithm. The details of our approach will be presented in a future paper.

We had great success applying our new reduced-set method to binary SVMs, for example reducing our binary cloud mask over land from 2456 support vectors to exactly 20, with essentially no loss in overall accuracy. We are still researching the best way to reduce a multi-class SVM. One straightforward technique that we have used successfully is to reduce each of the individual binary SVMs in a one-vs-all classifier, allocating a different number of support vectors to each binary classifier based on its difficulty. Specifically, we build each reduced binary SVM up one support vector at a time, each round adding a new support vector to the SVM with the lowest accuracy (relative to the unreduced SVM). We stop when the overall multi-class SVM has acceptible accuracy. This technique has yielded a new multi-class SVM with a total of 300 support vectors and 84.5% accuracy, relative to 85.5% accuracy for the unreduced SVM. In the future, we hope to develop methods that "share" support vectors between different binary SVMs within the multi-class classifier for possibly even greater reductions. However, our current techniques have given us SVMs that are at least in the right order of magnitude that we can experiment with running them in the operational MISR software framework.

6 Conclusions and Future Work

We have investigated the use of SVMs to perform cloud-type classification of individual pixels in MISR data. Using a combination of radiance and stereo height information and a large amount of training data, we were able to train a classifier that could classify over 85% of the pixels correctly, relative to human expert labels. Additionally, reduced-set techniques were developed and applied to yield more efficient SVMs with similiar accuracy, but much faster runtime for use in an operational setting.

We hope that this research helps open the door for future use of SVMs in remote-sensing pixel classification problems. While neural networks, decision trees, genetic algorithms, and several other techniques have been popular in such problems for years, support vector machines have many advantages, and now that it is possible to use them with no associated speed penalty, they should be considered more often.

Our immediate plans are to further develop our reduced-set techniques for multi-class support vector machines, continue to refine our cloud classifier with more training examples and eventually turn a final version of the classifier into an operational product. In addition, we intend to explore incorporating texture features, which are likely to help distinguish between certain classes of clouds.

7 Acknowledgements

We would like to thank Dave Diner and Rebecca Castaño for their support of this work. The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The MISR data used here were obtained from the NASA Langley Research Center Atmospheric Sciences Data Center.

References

1. Rossow, W.B., Schiffer, R.: ISCCP cloud data products. Bulletin of the American Meteorological Society **72** (1991) 2–20
2. Hahn, C.J., Rossow, W.B., Warren, S.G.: ISCCP cloud properties associated with standard cloud types identified in individual surface observations. Journal of Climate **14** (2001) 11–28
3. Bankert, R.: Naval research laboratory monterey GOES cloud classification (website) (2005) `http://www.nrlmry.navy.mil/sat-bin/clouds.cgi`.
4. Tag, P.M., Bankert, R.L., Brody, L.R.: An AVHRR multiple cloud-type classification package. Journal of Applied Meteorology **39** (2000) 125 – 134
5. Welch, R., Sengupta, S., Goroch, A., Rabindra, P., Rangaraj, N., Navar, M.: Polar cloud and surface classification using AVHRR imagery: An intercomparison of methods. Journal of Applied Meteorology **31** (1992) 405–420
6. Bankert, R.L.: Cloud pattern identification as part of an automated image analysis. In: Preprints, 7th American Meteorological Society Conference on Satellite Meteorology and Oceanography, Boston, MA (1994) 441–443
7. Bankert, R.L.: Cloud classification of AVHRR imagery in maritime regions using a probabilistic neural network. Journal of Applied Meteorology **33** (1994) 909–918
8. Bankert, R.L., Aha, D.W.: Automated identification of cloud patterns in satellite imagery. In: Preprints, 14th Conference on Weather Analysis and Forecasting, American Meteorological Society, Dallas, TX (1995) 313–316
9. Tian, B., Azimi-Sadjadi, M.R., Vonder Haar, T.H., Reinke, D.: Temporal updating scheme for probablistic neural network with application to satellite cloud classification. IEEE Transactions on Neural Networks **11** (2000) 903–920
10. Saitwal, K., Azimi-Sadjadi, M.R., Reinke, D.: A multichannel temporally adaptive system for continuous cloud classification from satellite imagery. IEEE Transactions on Geoscience and Remote Sensing **41** (2003) 1098 – 1104
11. Baum, B.A., Tovinkere, V., Titlow, J., Welch, R.: Automated cloud classification of global AVHRR data using a fuzzy logic approach. Journal of Applied Meteorology **36** (1997) 1519–1540
12. Azimi-Sadjadi, M.R., Zekavat, S.A.: Cloud classification using support vector machines. In: Proceedings of the 2000 IEEE Geoscience and Remote Sensing Symposium (IGRASS 2000). Volume 2., Honolulu, HI (2000) 669–671
13. Lee, Y., Wahba, G., Ackerman, S.: Cloud classification of satellite radiance data by multicategory support vector machines. Journal of Atmospheric and Oceanic Technology **21** (2004) 159–169
14. Li, J., Menzel, W.P., Yang, Z., Frey, R.A., Ackerman, S.A.: High-spatial-resolution surface and cloud-type classification from MODIS multispectral band measurements. Journal of Applied Meteorology **42** (2003) 204–226
15. Cortes, C., Vapnik, V.: Support-vector network. Machine Learning **20** (1995) 273–297
16. Burges, C.J.C.: Simplified support vector decision rules. In: Proceedings of the Thirteenth International Conference on Machine Learning. (1996) 71–77
17. Weston, J., Watkins, C.: Support vector machines for multiclass pattern recognition. In: Proceedings of the Seventh European Symposium On Artificial Neural Networks. (1999)
18. Hsu, C.W., Lin, C.J.: A comparison of methods for multi-class support vector machines. IEEE Transactions on Neural Networks **13** (2002) 415–425
19. Moroney, C., Muller, J.P., Davies, R.: Operational retrieval of cloud-top heights using MISR data. IEEE Transactions on Geoscience and Remote Sensing **40** (2002) 1532 – 1540
20. Horváth, A., Davies, R.: Simultaneous retrieval of cloud motion and height from polar-orbiter multiangle measurements. Geophysical Research Letters **28** (2001) 2915 – 2918
21. Garay, M.J., Mazzoni, D.M., Davies, R., Diner, D.: The application of support vector machines to the analysis of global datasets from MISR. In: Proceedings of the Fourth Conference on Artificial Intelligence Applications to Environmental Science, San Diego, CA (2005)
22. DeCoste, D., Schölkopf, B.: Training invariant support vector machines. Machine Learning **46** (2002)
23. DeCoste, D., Mazzoni, D.: Fast query-optimized kernel machine classification via incremental approximate nearest support vectors. In: Proceedings of the Twentieth International Conference on Machine Learning, Washington, DC (2003)
24. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11** (1997) 341–359

Canonical Correlation, an Approximation, and the Prediction of Protein Abundance

Anthony Bonner[*] Han Liu[†]

Abstract

This paper addresses a central problem of Bioinformatics and Proteomics: estimating the amounts of each of the thousands of proteins in a cell culture or tissue sample. Although laboratory methods involving isotopes have been developed for this problem, we seek a simpler method, one that uses fewer laboratory procedures. Specifically, our aim is to use data-mining methods to infer protein levels from the relatively cheap and abundant data available from high-throughput tandem mass spectrometry (MS/MS). In this paper, we develop and evaluate a method for tackling this problem. The method is based on a simple generative model of MS/MS data. We first show how to linearize the model and fit it to data using Canonical Correlation Analysis (CCA). Then, because CCA is computationally expensive for the large datasets we are dealing with, we develop an efficient approximation of CCA, one that exploits the structure of our data. We prove that the method is correct in that it achieves a well-defined optimization criterion. We also evaluate the method on several biological datasets. The datasets themselves were generated by MS/MS experiments performed on various tissue samples taken from Mouse.

keywords: Bioinformatics, Proteomics, Data Mining, Machine Learning, Peptides, Tandem Mass Spectrometry.

1 Introduction

Proteomics is the large-scale study of the thousands of proteins in a cell [9]. In a typical Proteomics experiment, the goal might be to compare the proteins present in a certain tissue under different conditions. For instance, a biologist might want to study cancer by comparing the proteins in a cancerous liver to the proteins in a healthy liver. Modern mass spectrometry makes this possible by enabling the identification of thousands of proteins in a complex mixture [13, 3]. However, *identifying* proteins is only part of the story. It is also important to *quantify* them, that is, to estimate how much of each protein is present in a cell [1, 5]. To this end, a number of laboratory methods have been developed, notably those based on mass tagging with isotopes [4, 12]. However, simpler, more-direct methods may be possible, methods that do not require additional laboratory procedures, but which are simply based on the data provided by tandem mass spectrometers [10]. This paper is an initial exploration of this possibility. In particular, we investigate the use of data-mining techniques to infer protein quantity from tandem mass spectrometry data.

1.1 Tandem Mass Spectrometry Tandem mass spectrometry involves several phases in which proteins are broken up and the pieces separated by mass [9, 13]. First, a complex mixture of thousands of unknown proteins is extracted from a cell culture or tissue sample. Since proteins themselves are too large to deal with, they are fragmented, producing a mixture of tens of thousands of unknown peptides. The peptides are then ionized and passed through a mass spectrometer. This produces a mass spectrum in which each spectral peak corresponds to a peptide. From this spectrum, individual peptides are selected for further analysis. Each such peptide is further fragmented and passed through a second mass spectrometer, to produce a so-called tandem mass spectrum. The result is a collection of tandem mass spectra, each corresponding to a peptide. Each tandem mass spectrum acts as a kind of fingerprint, identifying the peptide from which it came. By searching a database of proteins, it is possible to identify the protein that produced the peptide that produced the tandem mass spectrum. In this way, the proteins in the original tissue sample are identified. Often, the entire process is completely automatic.

A peptide mixture is not analyzed all at once. Instead, to increase sensitivity, the peptides are "smeared

---

[*] Department of Computer Science, University of Toronto. www.cs.toronto.edu/~bonner

[†] Department of Computer Science, University of Toronto. Mr. Liu was supported by the University of Toronto.

out" over time (often using liquid chromatography), so that different kinds of peptides enter the mass spectrometer at different times. A typical MS/MS experiment may last many hours, with proteins and peptides being identified each second. Copies of a particular peptide may continue to enter the mass spectrometer for several seconds or minutes. As the copies enter, the peptide will be repeatedly identified, once a second. In this way, a peptide may be identified and re-identified many times, increasing the confidence that the identification is correct. The number of times a particular peptide is identified is called the peptide's *spectral count*, since each identification requires the generation of a tandem mass spectrum. A large spectral count indicates that the peptide has been confidently identified.

In general, as protein abundance increases, so does spectral count [10]. However, the exact relationship is not at all clear and seems to depend on many factors, including the amino acid sequence of the peptides and the properties of the experimental set up. At present, there is no complete quantitative theory relating a protein's abundance to the spectral counts of its peptides. This paper is an initial attempt at using data-mining methods to develop such a theory, and using the theory to estimate protein abundance.

1.2 Data Mining To this end, the Emili Laboratory at the Banting and Best Department of Medical Research at the University of Toronto has provided us with datasets of several thousand proteins and peptides. The datasets were derived from MS/MS experiments on protein mixtures extracted from various tissue samples of Mouse. Each mixture contains tens of thousands of proteins, and each protein is present in the mixture with a specific (but unknown) abundance. A small sample of the data is shown in Table 1. (Details on how this data was generated can be found in [8].) Each row in the table represents a peptide ion. The first (left-most) column is the Swissprot accession number identifying a protein. The second column is the amino-acid sequence of the peptide. The third column is the spectral count of the peptide, and the last column is its charge. Notice that there may be many entries for the same protein, since a single protein can produce many peptides, and each peptide can produce ions with different amounts of charge. Protein ID, Peptide and Charge define a key for the table, that is, they uniquely identify a row.

High-throughput MS/MS experiments can provide a large amount of data of this kind on which to train and test data-mining methods. However, they also introduce a complication, since the amount of protein input to the mass spectrometer is unknown. This can be seen in Table 1, where spectral count is provided, but protein abundance is not. Thus, it is in general unclear whether a low spectral count for a peptide is due to the properties of the peptide or to a small amount of protein at the input. One of the challenges is to untangle these two influences. What makes the problem approachable is that we have data on spectral counts for peptides from the *same* protein, so differences in their counts cannot be due to differences in protein abundance. The data-mining methods developed and tested in this paper were chosen, in part, because of their ability to exploit this information. In effect, they treat protein abundance as a latent, or hidden variable, whose value must be estimated. In addition, they lead to efficient algorithms based on well-developed operators of linear algebra (specifically, eigenvector decomposition).

Table 1: A fragment of a data file

Protein ID	Peptide	Count	Charge
Q91VA7	$TRHNNLVIIR$	4	2
Q91VA7	$KLDLFAVHVK$	3	2
...	...	...	...

The methods are based on a simple generative model of MS/MS data. Because this is an initial study, we chose the model for its simplicity and tractability, and the goal was to see how well (or poorly) it fits the data, and to quantify the error. The model predicts the spectral count of a peptide based on two factors: its amino-acid sequence, and the abundance of the protein from which it was derived. We show that the model provides an explanation for the linear relationship between protein abundance and spectral count observed in [10]. More importantly, we show how to use the model to estimate protein abundance from spectral count.

Although the model is non-linear in the unknowns, it can be linearized without difficulty (Section 2.3). In its linearized form, the model can be fit to data using Canonical Correlation Analysis (CCA), a well-known statistical procedure that measures the linear relationship between two random vectors [7]. However, while we have found that CCA is quite adequate for small datasets, we have also found that it is computationally expensive for the large datasets we are dealing with. As an alternative, this paper develops an efficient algorithm for an approximation of CCA, one that avoids the need to deal with large, dense matrices. We show that the algorithm is correct for data of a certain form, which includes the kind of data we are dealing with. Finally, we evaluate the method on the real-world datasets provided by the Emili Laboratory, and compare its performance to CCA.

The rest of this paper is organized as follows. Section 2 shows how to use CCA to estimate protein quantity. Section 3 discusses the computational bottlenecks of CCA and develops our approximate method. Section 4 describes our experiments on real-world data, and presents and discusses our experimental results. Finally, Section 5 presents conclusions and suggests possible extensions. In addition, the appendix provides a proof that our approximation method is correct, *i.e.*, that it computes a well-defined approximate optimization criterion.

2 Using CCA to Mine MS/MS Data

This section first reviews Canonical Correlation Analysis (CCA), then presents our model of MS/MS data, and finally shows how to use CCA to fit the model to experimental data.

2.1 Canonical Correlation Analysis (CCA) Canonical correlation analysis (CCA), first developed by Hotelling [7], is a way of measuring the linear relationship between two multidimensional random variables, $\mathbf{X}$ and $\mathbf{Y}$. In this paper, we are interested in finding the largest of the so-called canonical correlations. This can be defined as finding a linear combination of the $\mathbf{X}$ variables and a linear combination of the $\mathbf{Y}$ variables that are maximally correlated. More formally, if we treat $\mathbf{X}$ and $\mathbf{Y}$ as column vectors, then we want to find two other column vectors, α and β, such that the correlation coefficient between $x = \mathbf{X}^T\alpha$ and $y = \mathbf{Y}^T\beta$ has maximal magnitude. We therefore want to maximize the magnitude of the following expression:

$$\begin{aligned} \rho &= \frac{E[(x-Ex)(y-Ey)]}{\sqrt{E(x-Ex)^2\, E(y-Ey)^2}} \\ &= \frac{E[\alpha^T(\mathbf{X}-E\mathbf{X})(\mathbf{Y}-E\mathbf{Y})^T\beta]}{\sqrt{E[\alpha^T(\mathbf{X}-E\mathbf{X})]^2\, E[(\mathbf{Y}-E\mathbf{Y})^T\beta]^2}} \qquad (2.1) \\ &= \frac{\alpha^T\mathbf{C}_{xy}\beta}{\sqrt{\alpha^T\mathbf{C}_{xx}\alpha\beta^T\mathbf{C}_{yy}\beta}} \end{aligned}$$

Here, $\mathbf{C}_{xy} = E[(\mathbf{X} - E\mathbf{X})(\mathbf{Y} - E\mathbf{Y})^T]$ is the covariance matrix of $\mathbf{X}$ and $\mathbf{Y}$.

Maximizing this expression leads to the following generalized eigenvalue equations:

$$(2.2) \qquad \begin{cases} \rho^2\mathbf{C}_{xx}\beta = \mathbf{C}_{xy}\mathbf{C}_{yy}^{-1}\mathbf{C}_{yx}\beta \\ \rho^2\mathbf{C}_{yy}\alpha = \mathbf{C}_{yx}\mathbf{C}_{xx}^{-1}\mathbf{C}_{xy}\alpha \end{cases}$$

The eigenvalue, ρ^2, is the square of the correlation coefficient whose magnitude we want to maximize. We should therefore choose the eigenvectors, α and β, with the largest eigenvalue. In other applications, eigenvectors and eigenvalues other than the maximum can be of interest. In general, the eigenvalues of these equations are known as the squared *canonical correlations* of $\mathbf{X}$ and $\mathbf{Y}$, and the eigenvectors are the canonical correlation *basis vectors.*

2.2 Modeling Spectral Counts This section presents our model of MS/MS data. The model represents a hypothesis about the way MS/MS data is generated. As mentioned above, because this is an initial study, the model was chosen largely for its simplicity and computational tractability. Section 4 evaluates the model on real MS/MS data.

To keep track of different proteins and peptides, we use two sets of indices, usually i for proteins and j for peptides. Proteins are numbered from 1 to N, and the peptides for the i^{th} protein are numbered from 1 to n_i. In addition, we use y to denote spectral count, and z to denote protein abundance. Each protein has a unique abundance, and each peptide has a unique spectral count. We therefore use z_i to denote the abundance of protein i, and y_{ij} to denote the spectral count of peptide j of protein i. With this notation, the following equation provides a simple model of spectral count:

$$(2.3) \qquad y_{ij} \;=\; z_i \cdot e_{ij}$$

This equation divides spectral count into two factors: z_i, the amount of protein from which peptide ij was generated; and e_{ij}, the *ionization efficiency* of the peptide. Ionization efficiency can be thought of as the propensity of the peptide to ionize and contribute to a peak, though it includes all factors that contribute to spectral count *other than* the amount of protein. In this way, we hope to untangle the amount of protein (which we want to estimate) from all other factors. Note that y_{ij} is observed, while z_i and e_{ij} are both unknown.

Of course, Equation 2.3 is not exact. It provides at best an approximate description of the data, and it is not yet clear what the errors look like. The rest of this paper spells out this model in greater detail, fits it to real-world data, quantifies the error, and estimates values for z_i and e_{ij} in the process.

It is worth noting that the model already accounts for an experimentally observed property of MS/MS data. Specifically, the abundance of a protein is directly proportional to the total spectral count of its peptides [10]. Formally,

$$z_i \;=\; b_i \sum_j y_{ij}$$

where b_i is an (unknown) proportionality constant that depends on the protein. The notion of ionization efficiency provides an explanation for this proportionality and a way of computing the constants b_i. In particular, it follows immediately from Equation 2.3 that

$$z_i \;=\; \frac{\sum_j y_{ij}}{\sum_j e_{ij}}$$

In other words, $b_i = 1/\sum_j e_{ij}$. Thus, according to the model of Equation 2.3, learning ionization efficiencies, e_{ij}, is the central problem in estimating protein abundance.

It should also be noted that with the model and data described above, we can only learn *relative* values of protein abundance and ionization efficiency, not absolute values. This is because any solution to Equation 2.3 is only unique up to a constant: multiplying all the z_i by a constant, and dividing all the e_{ij} by the same constant gives another, equally good solution. However, inferring relative protein abundance would be an extremely useful biological result. Moreover, by using a small amount of calibration data, the relative values can all be converted to absolute values.

In order to estimate relative values for these unknowns, we need a model of ionization efficiency. In this paper, we investigate the use of linear models, that is, models of the following form:

$$e_{ij} \;=\; x_{ij} \bullet \beta \tag{2.4}$$

Here, β is a vector of parameters (to be learned), x_{ij} is a vector of (known) peptide properties, and $\bullet$ denotes the dot product (or inner product) of the two vectors. The peptide properties could include such things as length, mass, charge, amino-acid composition, and estimates of various biochemical properties such as hydrophobicity. Section 4 spells out the specific properties used in this study.

Combining Equations 2.3 and 2.4 gives

$$y_{ij} \;=\; z_i \cdot (x_{ij} \bullet \beta) \tag{2.5}$$

Here, β is a parameter vector of our model, and z_i, the amount of protein, is a variable. Since the amounts of protein are unknown, each z_i is a latent, or hidden variable, whose values must be estimated. Note that if the values of z_i were known (*i.e.*, if they were included in the training data), then estimating a value for β would be a straightforward problem of multivariate linear regression. Unfortunately, the training data does *not* include this information, as can be seen in Table 1. This makes the model non-linear in the unknowns.

Fortunately, it is not hard to transform this non-linear model into a linear one. We simply divide both sides of the Equation 2.5 by z_i, to give the following linear equations:

$$y_{ij} \cdot \alpha_i \;=\; x_{ij} \bullet \beta \tag{2.6}$$

where $\alpha_i = 1/z_i$ is unknown. This, then, is our final model. Like Equation 2.3, it is an approximation, and our goal is to see how closely we can fit it to the data. Since the model is linear, we can use linear fitting methods, such as CCA.

2.3 Fitting the Model with CCA Since both sides of Equation 2.6 contain unknowns, we cannot estimate their values by minimizing the total error, as in linear regression. This is because the error is trivially minimized to 0 by setting $\alpha_i = 0$ and $\beta = 0$, which is clearly incorrect. However, we can choose values for α_i and β that maximize the correlation coefficient between the two sides of the equation. This is what CCA does.

To apply CCA we first express the problem in matrix notation. To this end, we construct the following sparse $N \times M$ matrix, $\mathbf{Y}$:

$$\left(\begin{array}{ccc|ccc|c|cc}
y_{11} & \cdots & y_{1n_1} & 0 & \cdots & 0 & \cdots & 0 & \cdots \\
0 & \cdots & 0 & y_{21} & \cdots & y_{2n_2} & \cdots & 0 & \cdots \\
 & \cdots & & & \cdots & & & & \cdots \\
0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots \\
0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & y_{N1} & \cdots
\end{array}\right)$$

where N is the number of proteins, $M = \sum_i n_i$ is the total number of peptides, and y_{ij} is the spectral count of peptide j of protein i. We also construct the following $p \times M$ matrix:

$$\mathbf{X} \;=\; (x_{11}, ..., x_{1n_1}, x_{21}, ..., x_{2n_2}, ..., x_{N1}, ..., x_{Nn_N})$$

where each x_{ij} is a column vector of length p, the vector of peptide properties defined above. Finally, in addition to these two (known) matrices, $\mathbf{X}$ and $\mathbf{Y}$, we define a new (unknown) column vector, $\alpha = (\alpha_1, ..., \alpha_N)^T$, where $\alpha_i = 1/z_i$. With these definitions, Equation 2.6 can be rewritten as:

$$\mathbf{Y}^T \alpha \;=\; \mathbf{X}^T \beta \tag{2.7}$$

Our aim is now to estimate values for α and β. From the value of α, we can easily estimate an input amount for each protein, using $\hat{z}_i = 1/\alpha_i$; and from the value of β, we can easily estimate an ionization efficiency for any peptide, using $\hat{e}_{ij} = \mathbf{x}_{ij} \bullet \beta$.

We use CCA to find values for α and β that maximize the sample correlation coefficient between the two random vectors $\mathbf{Y}^T\alpha$ and $\mathbf{X}^T\beta$. From Equations 2.2, we need the sample covariance matrices of $\mathbf{X}$ and $\mathbf{Y}$. These are given by the following matrix equations:

$$\begin{array}{rcl} \mathbf{C}_{xx} & = & (\mathbf{X} - \overline{\mathbf{X}})(\mathbf{X} - \overline{\mathbf{X}})^T \\ \mathbf{C}_{yy} & = & (\mathbf{Y} - \overline{\mathbf{Y}})(\mathbf{Y} - \overline{\mathbf{Y}})^T \\ \mathbf{C}_{xy} & = & (\mathbf{X} - \overline{\mathbf{X}})(\mathbf{Y} - \overline{\mathbf{Y}})^T \\ \mathbf{C}_{yx} & = & (\mathbf{Y} - \overline{\mathbf{Y}})(\mathbf{X} - \overline{\mathbf{X}})^T \end{array}$$

Here, $\overline{\mathbf{X}}$ is a matrix of sample means. That is, element ij in matrix $\overline{\mathbf{X}}$ is the average of all the elements in row i of matrix $\mathbf{X}$ (*i.e.*, the sample mean of peptide feature i). Likewise for $\overline{\mathbf{Y}}$. With these covariance matrices, Equations 2.2 give the values of α and β that maximize the correlation coefficient.

Although straightforward, we have found that solving Equations 2.2 is computationally expensive, both in terms of time and space. The main problem is the large size of the matrix $\mathbf{Y}$. The data we are dealing with contains roughly 10,000 different peptides and 2,000 different proteins. Matrix $\mathbf{Y}$ therefore has dimensions $2,000 \times 10,000$, and so the covariance matrix $\mathbf{C}_{yy}$ has dimensions $2,000 \times 2,000$. The first of Equations 2.2 requires inverting this matrix. However, inverting such a large matrix requires considerable time and space, and can lead to severe numerical problems [14]. The second of Equations 2.2 requires inverting the covariance matrix $\mathbf{C}_{xx}$, which is not nearly so large and can easily be inverted. However, on the left side of the equation, we again encounter the large matrix $\mathbf{C}_{yy}$, which makes the generalized eigenvector equation costly to solve. In addition, since $\mathbf{C}_{yy}$ is a $N \times N$ matrix, where N is the number of proteins, the cost of this method increases rapidly with the number of proteins in the dataset.

3 An Efficient Approximation

Although it is very large, the matrix $\mathbf{Y}$ defined in the previous section is also very sparse. In every column, only one element is non-zero. By exploiting the structure and sparseness of this matrix, we develop an efficient algorithm for an approximation of CCA.

In CCA, the statistical measure of similarity between two random vectors is *correlation coefficient.* Formally, we want to find α and β that maximize the following expression:

$$\frac{(\mathbf{Y}^T\alpha - \overline{\mathbf{Y}^T\alpha}) \bullet (\mathbf{X}^T\beta - \overline{\mathbf{X}^T\beta})}{\|\mathbf{Y}^T\alpha - \overline{\mathbf{Y}^T\alpha}\| \cdot \|\mathbf{X}^T\beta - \overline{\mathbf{X}^T\beta}\|} \tag{3.8}$$

The point to notice here is that the two vectors are centered, by subtracting their means. The correlation coefficient is thus the cosine of the angle between the two centered vectors, and CCA finds α and β to minimize this angle. Unfortunately, from a computational point of view, the centering of the vectors causes a great deal of problems, because it effectively destroys the sparse structure of matrix $\mathbf{Y}$. This is because the large covariance matrix, $\mathbf{C}_{yy}$, is defined not in terms of $\mathbf{Y}$, but in terms of $\mathbf{Y} - \overline{\mathbf{Y}}$. Unfortunately, although $\mathbf{Y}$ is sparse, $\overline{\mathbf{Y}}$ is dense, so $\mathbf{Y} - \overline{\mathbf{Y}}$ is also dense. In fact, in row i of matrix $\overline{\mathbf{Y}}$, each entry is $\sum_{j=1}^{n_i} y_{ij}/N$, so $\overline{\mathbf{Y}}$ has no zeros and is maximally dense. Likewise for $\mathbf{Y} - \overline{\mathbf{Y}}$.

In the approximation method developed here, we retain the idea of minimizing the angle, but without the requirement of centering the vectors first. That is, we minimize the angle between the uncentered vectors $\mathbf{Y}^T\alpha$ and $\mathbf{X}^T\beta$, by maximizing the cosine of the angle between them. This amounts to maximizing the following expression:

$$\frac{\mathbf{Y}^T\alpha \bullet \mathbf{X}^T\beta}{\|\mathbf{Y}^T\alpha\| \cdot \|\mathbf{X}^T\beta\|} \tag{3.9}$$

Maximizing this expression leads to the same generalized eigenvector equations given in Equation 2.2, except that now the covariance matrices are defined in terms of uncentered random variables. Thus, instead of $\mathbf{C}_{yy} = (\mathbf{Y} - \overline{\mathbf{Y}})(\mathbf{Y} - \overline{\mathbf{Y}})^T$, we now use $\mathbf{C}_{yy} = \mathbf{Y}\mathbf{Y}^T$.

Of course, this does not change the dimensions of any of the covariance matrices. In particular, $\mathbf{C}_{yy}$ is still very large. However, it is now possible to simplify Equations 2.2 so that the remaining matrices are relatively small. This is shown in Theorem 1 below. In this theorem, Y_i is the column vector $(y_{i1}, y_{i2}, ..., y_{in_i})^T$, and $\mathbf{X}_i$ is the matrix $(x_{i1}, x_{i2}, ..., x_{in_i})$. They represent, respectively, the spectral counts and peptide properties for protein i. Note that $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_N)$, so each $\mathbf{X}_i$ is a vertical slice of the larger matrix $\mathbf{X}$.

Theorem 1: *Expression 3.9 above is maximized when the parameter vector β is a solution of the following generalized eigenvector equation:*

$$\rho^2 \boldsymbol{X}\boldsymbol{X}^T \beta \quad = \quad [\sum_i \boldsymbol{X}_i Y_i Y_i^T \boldsymbol{X}_i^T / \|Y_i\|^2]\,\beta \tag{3.10}$$

Moreover, it is the eigenvector with the largest eigenvalue, ρ^2. In addition, $\rho = \cos\theta$, where θ is the angle *between the vectors $\boldsymbol{Y}^T\alpha$ and $\boldsymbol{X}^T\beta$. Finally, the elements of the parameter vector α are given by the following equation:*

$$\alpha_i \;=\; \boldsymbol{Y}_i^T \boldsymbol{X}_i^T \beta / \rho \|Y_i\|^2 \tag{3.11}$$

Proof: Given in the appendix.

Comparing Equation 3.10 in this theorem with Equation 2.2 in Section 2.1, the important point is that we no longer need to compute or invert the large covariance matrix $\mathbf{C}_{yy}$. As for the other matrices, $\mathbf{X}_i$ has dimensions $p \times n_i$, where p is the number of properties (or features) characterizing each peptide. Matrix $\mathbf{X}$ has dimensions $p \times M$, where $M = \sum_i n_i$ is the total number of peptides. The matrices $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}_i Y_i Y_i^T \mathbf{X}_i^T$ thus have dimensions $p \times p$. It is the size of these $p \times p$ matrices that is important, since this determines the cost of solving eigenvector equation 3.10.

Since our datasets contain roughly 10,000 peptides, M is large. However, p, the number of peptide features is much smaller. In this paper, we use two different feature sets, one with $p = 21$ and one with $p = 421$, as described in Section 4. With $p = 21$ the eigenvector matrices are very small. With $p = 421$ they are much larger, and although they slowed down the eigenvector computations discernably, they still posed no significant problems. Moreover, they are considerably smaller than the covariance matrix $\mathbf{C}_{yy}$ in Equation 2.2, whose dimensions are roughly 2000×2000, which did cause significant computational problems. Perhaps more importantly, the size of the $p \times p$ matrices is independent of how many proteins or peptides are in the dataset. Thus, computational cost will not be significantly affected as the datasets grow.

3.1 Weighting the Data

In fitting a model to data, one may not wish to treat all data points equally, but to place different importance on different data. To allow for this, one can introduce weights into the optimization criterion. For our approximation to CCA, the optimization criterion is given by Equation 3.9, which specifies the angle between two vectors. In this case, we can define what might be called a generalized angle, in which different vector components are weighted differently. If the two vectors are denoted U and V, then the generalized angle is defined by (the arc cosine of) the following expression:

$$\frac{U^T \mathbf{W} V}{\sqrt{U^T \mathbf{W} U}\,\sqrt{V^T \mathbf{W} V}}$$

Here, $\mathbf{W}$ is a diagonal matrix, whose i^{th} diagonal element, $\mathbf{w}_i$, is the weight of the i^{th} component of the vectors. When $\mathbf{W}$ is the identity matrix, we get the ordinary, unweighted angle. To compute the generalized angle between U and V, we can use the above formula, or we can first transform the vectors as follows:

$$U_i' \;=\; U_i \sqrt{\mathbf{w}_i} \qquad\qquad V_i' \;=\; V_i \sqrt{\mathbf{w}_i}$$

and then compute the unweighted angle between U' and V'. This latter approach shows that giving weight $\mathbf{w}_i$ to data point (U_i, V_i) is equivalent to simply multiplying U_i and V_i by $\sqrt{\mathbf{w}_i}$. This makes intuitive sense, since the angle between two vectors is more strongly influenced by large vector components than by small ones.

In the case of our MS/MS data, this corresponds to assigning a different weight to each peptide and transforming its spectral count and feature vector as follows:

$$y_{ij}' \;=\; y_{ij} \sqrt{\mathbf{w}_{ij}} \qquad\qquad x_{ij}' \;=\; x_{ij} \sqrt{\mathbf{w}_{ij}}$$

where $\mathbf{w}_{ij}$ is the weight assigned to peptide j of protein i. We then apply Theorem 1 to y_{ij}' and x_{ij}', instead of to y_{ij} and x_{ij}. The choice of what weights to use is heuristic, and in our experiments, we chose two different sets of weights, $\mathbf{w}_{ij} = ||Y_i||$ and $\mathbf{w}_{ij} = 1/||Y_i||$, respectively. These weights are a simple attempt to address two different sources of noise and error. The first set of weights emphasizes peptides from proteins with high spectral counts, since they have a higher reliability and a better signal-to-noise ratio. The second set of weights attempts to stabilize the model error, assuming that peptides from proteins with larger spectral counts will tend to have larger error.

4 Experiments

This section uses real-world data to experimentally evaluate the data-mining methods and models described above. The main evaluation strategy is ten-fold cross validation, with correlation coefficient used to measure the fit of a learned model to the testing portion of the data. The main difficulty in carrying out the evaluation was the distribution of the spectral counts, which ranges over several orders of magnitude and is highly skewed, with most data concentrated at very low values. To deal with this difficulty, we use the Spearman rank correlation coefficient to measure the goodness-of-fit [2]. Unlike the more common Pearson correlation coefficient, which measures *linear* correlation, Spearman's coefficient measures *monotone* correlation and is insensitive to extreme data values. In addition, we use plots of observed v.s. estimated values to provide an informative visualization of the fit.

4.1 Study Design We evaluated the data-mining methods on three datasets derived from tissue samples taken from Mouse. Similar in form to Table 1, the datasets were provided by the Emili Laboratory at the Banting and Best Department of Medical Research at the University of Toronto. We refer to these datasets as **Mouse Brain Data**, **Mouse Heart Data**, and **Mouse Kidney Data**. In each of these datasets, many proteins have multiple entries, each entry corresponding to a different peptide. In fact, the vast majority of entries correspond to proteins of this type. However, some proteins have only one entry, since they give rise to only one observable peptide. Such proteins provide no information about protein abundance, so we remove them from the datasets. After removal, the Brain dataset contains 8,527 peptides and 1,664 proteins, Heart dataset contains 7,660 peptides and 1,281 proteins, and the Kidney dataset contains 7,074 peptides and 1,291 proteins.

For the data-mining methods developed in this paper, each peptide must be represented as a vector, x. This section evaluates two ways of doing this, using vectors with 21 features and 421 features, respectively. The vectors with 21 features represent the amino-acid composition of a peptide. Since there are twenty different amino acids, the vector has 20 features, $(x_1, ..., x_{20})$, where the value of feature x_i is the number of occurrences of a particular amino acid in the peptide. In addition, the vector has a 21^{st} feature, x_0, whose value is always 1, to represent a bias term, as is common in data-mining and machine-learning models [6]. The vectors with 421 features include the original 21 plus an additional 400 features representing the dimer composition of a peptide. A *dimer* is a sequence of two amino acids, and since there are 20 distinct amino acids, there are 400 distinct dimers.[1]

We evaluated numerous combinations of feature vector, data-mining method and weighting scheme. Due to space limitations, we present only five of them here. In addition, because of the time required to execute CCA, we used it in only one combination: unweighted and with vectors having 21 features. We also evaluated four versions of the approximate method developed in Section 3. The first two versions are both unweighted and use vectors with 21 features and 421 features, respectively. We refer to these two versions as *Approx-21* and *Approx-421*. The other two versions are both weighted and use vectors with 21 features. The two weighting schemes used are $\mathbf{w}_i = ||Y_i||$ and $\mathbf{w}_i = 1/||Y_i||$, as described in Section 3.1.

Using ten-fold cross validation, we evaluated each of these five data-mining methods on each of the three Mouse datasets. Thus, each method was trained on nine tenths of the data (the training set), and the fitted model was then evaluated on the remaining one tenth of the data (the test set), and this was repeated in ten possible ways. Each training session produced an estimate, $\hat{\beta}$, of the parameter vector β, and an estimate, $\hat{z}$, of the input amount for each protein in the training set. Using $\hat{\beta}$, we estimated the ionization efficiency of each peptide in the entire dataset, using the formula $\hat{e} = x \bullet \hat{\beta}$, where x is the vector representation of the peptide. Applying univariate linear regression to Equation 2.3, we then estimated an input amount, $\hat{z}$, for each protein in the test set. We then estimated the spectral count of each peptide in the entire dataset, using $\hat{y} = \hat{z} \cdot \hat{e}$. Finally, we compared the estimated and observed spectral counts (that is, $\hat{y}$ and y) by computing Spearman rank correlation coefficients.

The results are shown in Table 2. In this table, each column corresponds to a Mouse dataset, and each row corresponds to a data-mining method. Each position in the table shows four numbers, stacked vertically. The top two numbers are the mean and standard deviation of the correlation coefficient of $\hat{y}$ and y on the training data. The bottom two numbers are the mean and standard deviation of the correlation coefficient on the test data. (Since at this stage, we are only interested in rough estimates of correlation coefficient, the ten estimates produced by ten-fold cross validation are enough.) In addition, by dividing $\hat{y}$ and y by $\hat{z}$, we get two different estimates of ionization efficiency, which we denote $\hat{e}$ and e, respectively. The correlation coefficient between these two estimates is what CCA tries to maximize. Thus, while the correlation coefficient of $\hat{y}$ and y measures the ability of the fitted model to predict experimental observations, the correlation coefficient of $\hat{e}$ and e provides the most direct measure of fit between the model and the data. The results are shown in Table 3, which has the same format as Table 2.

4.2 Results The first point to notice is that of all the methods that use vectors with 21 features, CCA provides the best fit to the training data in Table 3. (only Approx-421 produces a better fit, and only on the Kidney data, but it uses more features.) This is

---

[1] In [11] we explore other peptide features, including peptide charge.

Table 2: Correlation of y and $\hat{y}$ on real data

Method	Statistics	Brain Data	Heart Data	Kidney Data
CCA	Mean Train:	0.0350	0.0273	0.0335
	Std Train:	0.0292	0.0162	0.0413
	Mean Test:	0.3694	0.2575	0.3563
	Std Test:	0.1118	0.1683	0.0865
Approx -21	Mean Train:	0.2180	0.3319	0.2850
	Std Train:	0.0356	0.0135	0.0583
	Mean Test:	0.4190	0.4250	0.4109
	Std Test:	0.1133	0.0906	0.0705
Approx -421	Mean Train:	0.0660	0.1299	0.1742
	Std Train:	0.0529	0.1631	0.0745
	Mean Test:	0.2869	0.2839	0.4090
	Std Test:	0.1975	0.1098	0.0999
Weighted Approx $w = \|y\|$	Mean Train:	0.2121	0.4004	0.3518
	Std Train:	0.0678	0.0935	0.0363
	Mean Test:	0.4225	0.4406	0.4302
	Std Test:	0.1004	0.0934	0.0951
Weighted Approx $w = \|1/y\|$	Mean Train:	0.2568	0.3811	0.3005
	Std Train:	0.0067	0.0168	0.0186
	Mean Test:	0.3924	0.4223	0.3999
	Std Test:	0.0924	0.0799	0.0586

Table 3: Correlation of e and $\hat{e}$ on real data

Method	Statistics	Brain Data	Heart Data	Kidney Data
CCA	Mean Train:	0.6027	0.5929	0.5971
	Std Train:	0.0042	0.0054	0.0040
	Mean Test:	0.1054	0.0459	0.2049
	Std Test:	0.0763	0.0486	0.0846
Approx -21	Mean Train:	0.4298	0.3921	0.4078
	Std Train:	0.0057	0.0081	0.0074
	Mean Test:	0.2759	0.2234	0.2530
	Std Test:	0.0432	0.0605	0.0485
Approx -421	Mean Train:	0.5460	0.5469	0.6080
	Std Train:	0.1998	0.2616	0.0071
	Mean Test:	0.0982	0.0913	0.2335
	Std Test:	0.1125	0.1058	0.0424
Weighted Approx $w = \|y\|$	Mean Train:	0.4613	0.3909	0.3916
	Std Train:	0.0051	0.0104	0.0135
	Mean Test:	0.2021	0.1163	0.2272
	Std Test:	0.0737	0.0788	0.0456
Weighted Approx $w = \|1/y\|$	Mean Train:	0.3118	0.2995	0.3072
	Std Train:	0.0056	0.0070	0.0093
	Mean Test:	0.1999	0.1786	0.1905
	Std Test:	0.0349	0.0528	0.0376

to be expected since CCA maximizes the correlation coefficient, which is what the table measures. On the other hand, Approx-21 provides the best fit to the test data. The same is true in Table 2, where Approx-21 provides better test predictions of y, the spectral count (the main biological observable). These results suggest that while our method may be an approximation of CCA, it may also be more appropriate for this problem, in terms of accuracy as well as speed.

The effect of the weighted methods is inconclusive. In Table 3, the unweighted Approx-21 has consistently better performance on the test data than either of the two weighted schemes, but not dramatically better. In Table 2, the three methods perform comparably on the test data, though weights of $|1/y|$ seem to be marginally best, and weights of $|y|$ seems to be marginally worst, with Approx-21 in between.

The effect of the larger feature vector is more conclusive. If we compare the Approx-21 and Approx-421 methods in Table 3, we can see that Approx-421 shows evidence of overfitting, since the fit on the testing data is often much worse than on the training data. The 400 dimer values included among the 421 features thus appear to have little predictive value. Biologically, this a useful negative result.

Table 2 shows some apparently anomalous patterns. For instance, the fit on the testing data is often better than on the training data. Also, Approx-21 has a better fit to the training data than Approx-421, even though the features used in Approx-21 are a subset of those used in Approx-421. These patterns are probably a result of comparing $\hat{y}$ and y, whereas the data-mining methods try to fit $\hat{e}$ and e. The same patterns are not present in Table 3, which compares $\hat{e}$ and e.

In addition to the measurements presented in Tables 2 and 3, Figures 1 and 2 provide a visual representation of how well the estimated models fit the data. In each figure, the horizontal axis is $\hat{e}$, the vertical axis is e, and each point represents a single peptide. Figure 1 was generated by the Approx-21 method, and Figure 2 by the Approx-421 method, with both trained on the entire Kidney dataset. The other methods generate similar figures. The first point to notice is that in both figures, the vast majority of values of $\hat{e}$ and e are positive, which is how things should be, since ionization efficiency is inherently positive. The Approx-421 method has more negative points than Approx-21, but again, this could be a result of overfitting. The second point to notice is that each figure appears to consist of two components—a fairly linear diagonal component, and a less-linear horizontal blob. This suggests that there are two populations of peptides, those whose ionization efficiency is well-modeled by a linear function, and those whose ionization efficiency is much less predictable. This would explain why the correlation coefficients in Table 3 are low. It also suggests a natural topic for future research: characterizing those peptides that can be modeled linearly.

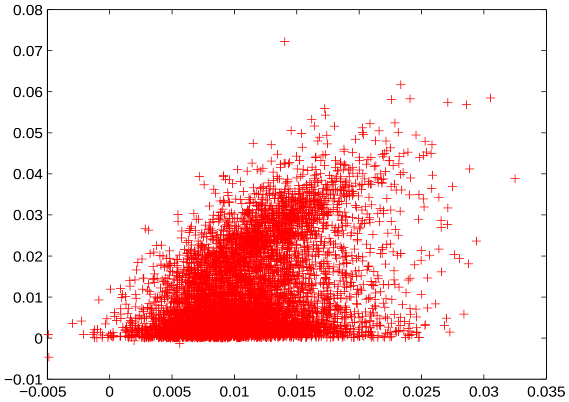


Figure 1: e vs. $\hat{e}$ as estimated by Approx-21.

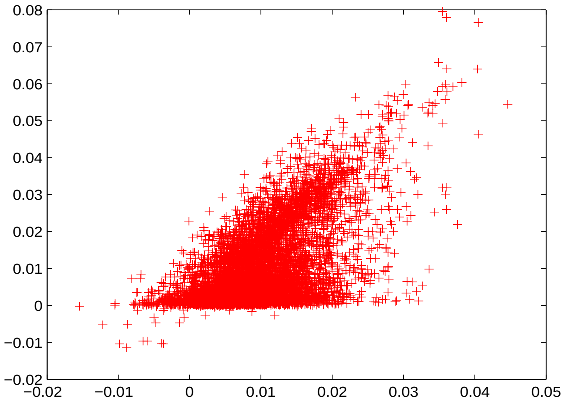


Figure 2: e vs. $\hat{e}$ as estimated by Approx-421.

5 Conclusions and Future Directions

This paper developed and evaluated a data-mining method for estimating protein levels from high-throughput Tandem Mass Spectrometry (MS/MS) data. The method is based on a simple generative model of MS/MS data. We showed how to linearize the model and fit it to data using Canonical Correlation Analysis (CCA). However, for the large data sets we are working with, we found CCA to be computationally expensive. As an alternative, we developed an efficient algorithm for an approximation to CCA, one that exploits the structure of our data.

We provided a proof of correctness of the method, and we evaluated its effectiveness on real MS/MS data derived from tissue samples of Mouse. The evaluations included three different tissue samples, two different vector representations of peptides, three different schemes for weighting the data, and three different evaluation measures (two based on correlation coefficients and one based on data visualization). The results suggest that our method may be better than CCA at fitting the model to MS/MS data. Biologically, they suggest that spectral count is not influenced by the dimers in a peptide. They also suggest that there may be two types of peptide, only one of whose ionization efficiency can be adequately modeled by a linear function.

This research is just a first step, and additional work is needed before protein levels can be predicted accurately. This includes developing non-linear models of ionization efficiency (*e.g.*, models based on regression trees, neural nets, or support vector machines), identifying those peptides for which linear models are adequate, and investigating additional peptide representations, especially ones that retain more of the sequential content of a peptide. Other possibilities include developing more sophisticated models of the entire MS/MS process, especially models that account for interactions between peptides in the mass spectrometer.

Finally, the methodology of Section 4 evaluates a model in terms of its ability to predict the spectral counts of peptides, based on peptide properties and predicted protein abundance. The ability to accurately predict spectral counts in this way would be strong evidence that a model is correct, and would suggest that protein abundance was accurately predicted. However, conclusive proof requires a more direct comparison with known protein amounts (*e.g.*, as measured by the more laboratory-intensive isotope marker experiments [4, 12]).

Acknowledgments

The authors would like to thank Dr. Andrew Emili for his comments and discussions and for generous access to his data. We would also like to thank Clement Chung for generating and preprocessing the data. Finally, we would like to thank the anonymous reviewers for their constructive and helpful comments.

References

[1] R. Aebersold, M. Mann. *Mass spectrometry-based proteomics*, Nature 422, pp 198-207, 2003.

[2] P. Bickel and K. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*, Holden-Day INC, 1977.

[3] Joshua E Elias, Francis D Gibbons, Oliver D King, Frederick Roth, Steven P Gygi *Intensity-based protein identification by machine learning from a library of tandem mass spectra*,Nature, biotechnology. Volume 22 Number 2, 2004

[4] S.P. Gygi, B. Rist, S.A. Gerber, F. Turecek, M.H. Gelb, and R. Aebersold. *Quantitative analysis of complex protein mixtures using isotope-coded affinity tags*, Nature Biotechnology 17, pp 994–999.

[5] S.P.Gygi and R. Aeberold. *Mass Spectrometry and Proteomics*, Current Opinion in Chemical Biology, 4, pp 489–494, 2000.

[6] T. Hastie, R. Tibshirani, J. Friedman. *The elements of statistical learning—Data mining, inference and prediction*, Springer 2001.

[7] H. Hotelling. *Relations between two sets of variates.* Biometrika, 28:321-377, 1936.

[8] T. Kislinger, K. Ramin, D. Radulovic, et al. *PRISM, a Generic Large Scale Proteomics Investigation Strategy for Mammals*, Molecular & Cellular Proteomics 2.1 2003.

[9] D.C. Liebler. *Introduction to Proteomics, tools for the new biology*, Humana Press, NJ, 2002.

[10] H. Liu, R.G. Sadygov, and J.R. Yates. *A Model for Random Sampling and Estimation of Relative Protein Abundance in Shotgun Proteomics*, Anal. Chem. 76, pp 4193–4201, 2004.

[11] H. Liu. *Development and Evaluation of Methods for Predicting Protein Levels from Tandem Mass Spectrometry Data*, Masters thesis, Department of Computer Science, University of Toronto. January 2005.

[12] S. Ong, B. Blagoev, I. Kratchmarovat, D.B. Kristensen, H. Steen, A. Pandey, and M. Mann. *Stable Isotope Labelling by Amino Acid in Cell Culture, SILAC, as a Simple and Accurate Approach to Expression Proteomics*, Molecular & Cellular Proteomics 1.5 2002.

[13] G. Siuzdak. *The Expanding Role of Mass Spectrometry in Biotechnology*, Mcc Press, 2003.

[14] David S. Watkins *Fundamentals of Matrix Computation*, Wiley-Interscience 2002.

6 Appendix: Proof of Theorem 1

Observe that the maximum of expression 3.9 is the same as the maximum of the simpler expression $\mathbf{Y}^T\alpha \bullet \mathbf{X}^T\beta$ subject to the constraints $\|\mathbf{Y}^T\alpha\| = 1$ and $\|\mathbf{X}^T\beta\| = 1$. In fact, α and β maximize the unconstrained expression if and only if α' and β' maximize the constrained expression, where $\alpha' = \alpha/\|\mathbf{X}^T\alpha\|$ and $\beta' = \beta/\|\mathbf{X}^T\beta\|$. It is therefore sufficient to maximize the simpler, constrained expression. To carry this out, we use Lagrange multipliers and maximize the following expression:

$$\mathbf{Y}^T\alpha \bullet \mathbf{X}^T\beta \;-\; \lambda(\|\mathbf{Y}^T\alpha\|^2 - 1) \;-\; \mu(\|\mathbf{X}^T\beta\|^2 - 1)$$

It is not hard to see that this expression is equivalent to the following:

$$\sum_i \alpha_i Y_i^T \mathbf{X}_i^T \beta \;-\; \lambda(\sum_i \|\alpha_i Y_i\|^2 - 1) \;-\; \mu(\|\mathbf{X}^T\beta\|^2 - 1)$$

Taking partial derivatives with respect to β and α_i and setting the results to 0 gives the following equations:

$$\sum_i \alpha_i \mathbf{X}_i Y_i \;=\; 2\mu \mathbf{X}\mathbf{X}^T\beta \tag{6.12}$$

$$Y_i^T \mathbf{X}_i^T \beta \;=\; 2\lambda\alpha_i \|Y_i\|^2 \tag{6.13}$$

Left-multiplying Equation 6.12 by β^T gives

$$\beta^T \sum_i \alpha_i \mathbf{X}_i Y_i \;=\; 2\mu\beta^T \mathbf{X}\mathbf{X}^T\beta \tag{6.14}$$

$$=\; 2\mu\|\mathbf{X}^T\beta\|^2 \tag{6.15}$$

$$=\; 2\mu \tag{6.16}$$

since, by our constraint, $\|\mathbf{X}^T\beta\| = 1$. In a similar fashion, multiplying Equation 6.13 by α_i and summing over i gives

$$\sum_i \alpha_i Y_i^T \mathbf{X}_i^T \beta \;=\; 2\lambda \sum_i \alpha_i^2 \|Y_i\|^2 \tag{6.17}$$

$$=\; 2\lambda\|\mathbf{Y}^T\alpha\|^2 \tag{6.18}$$

$$=\; 2\lambda \tag{6.19}$$

since, by our constraint, $\|\mathbf{Y}^T\alpha\| = 1$. Note that Equations 6.16 and 6.19 can be rewritten as follows:

$$2\lambda \;=\; \mathbf{Y}^T\alpha \bullet \mathbf{X}^T\beta \;=\; 2\mu \tag{6.20}$$

In other words, $\lambda = \mu = \rho/2$, where ρ is the value we are maximizing. From this and Equation 6.13, it follows immediately that

$$\alpha_i \;=\; Y_i^T \mathbf{X}_i^T \beta/\rho\|Y_i\|^2 \tag{6.21}$$

This proves Equation 3.11. To prove Equation 3.10, note that from Equations 6.12 and 6.21, we get

$$2\mu\rho\mathbf{X}\mathbf{X}^T\beta \;=\; [\sum_i \mathbf{X}_i Y_i Y_i^T \mathbf{X}_i^T/\|Y_i\|^2]\,\beta$$

The result follows immediately, since $2\mu\rho = \rho^2$.

LibFeature: A software library for quickly generating feature vectors on the fly from structured data

Dominic Mazzoni

Jet Propulsion Laboratory, California Institute of Technology,
Pasadena, CA 91109, USA
`Dominic.Mazzoni@jpl.nasa.gov`

Abstract. We have developed a software library, LibFeature, that greatly simplifies the task of extracting feature vectors from raw data. The instructions for computing feature vectors from the input data are written in a high-level language, which can be interpreted in real-time, but because the language is deterministic, it can be executed on many feature vectors in parallel, resulting in performance comparable to efficient C code. We describe the capabilities of LibFeature, the Feature Description Language (FDL), the internal architecture and optimizations, and then show some benchmarks of its performance, while using realistic examples of constructing features from scientific image and time-series data throughout.

1 Introduction

When applying a machine learning or data mining algorithm to a new data set, it is often the case that the majority of the researcher's time is spent writing code to parse the data set and manipulate it into a form that can be used by the algorithm. Many such algorithms expect data to be in the form of *feature vectors*, each of which contains all of the information that the algorithm has available about one input example. For scientific and engineering data sets, feature vectors are typically composed of all real numbers representing measurements such as the image intensity at a particular pixel, or the electrical current of an instrument at a particular time. Assuming that these values are already available in data files of a known format, various operations might need to be done in order to construct feature vectors from the raw data, including:

1. Combining data from multiple files or multiple tables, for example if each band of a multispectral imager is stored in a separate array
2. Changing the units or normalizing values so that all elements in each feature vector are approximately the same magnitude
3. Computing features based on mathematical functions of raw values, for example computing the Normalized Difference Vegetation Index (NDVI) given radiance at both red and near-infrared wavelengths
4. Converting from categorical features to numerical, for example a sensor that reads `OFF`, `READY`, or `ERROR` could be represented by the features 0, 1, and 2, or alternatively by the vectors $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$
5. Augmenting the feature vector for one particular time or location with features from neighboring examples, for context
6. Filtering out feature vectors when one of the features is missing or bogus, or alternatively, interpolating the value of missing features in that case

We have written a software library, called *LibFeature*, that attempts to make this process easier by allowing one to specify the commands to produce a feature vector in a high-level language. Because LibFeature is highly optimized and because of the inherent parallelizability of the problem, it is usually possible to use LibFeature without suffering any speed penalty for using the additional layer and the parsing of the high-level language. In fact, LibFeature is often within a factor of two of a straightforward C implementation, and thus it is fast enough to be used in near-real-time systems.

One of the major challenges in designing a software library to replace a programming task that is time-consuming but not difficult, is that the new software library must be especially clean, lightweight, portable, and easy to use, otherwise most people will find it easier to just reinvent the wheel each time rather than introduce a dependency on another library. LibFeature is written in very portable C and is designed to compile and run on almost any modern computing platform, including Windows, Mac OS X, Linux, and any modern Unix system. It is powerful enough to handle surprisingly complicated calculations, but the most common tasks are designed to be as easy as possible.

In this paper, we will describe an outline of the system and the major capabilities and limitations, introduce the feature description language, show some examples of many common tasks that can be accomplished using LibFeature, discuss how LibFeature works and some of the optimizations it uses in order to achieve good performance, and finally examine some benchmarks comparing its performance to C programs.

2 Related Work

While we are not aware of any previous work to develop a software library specifically for constructing feature vectors for machine learning and data mining use, nevertheless many other software programs and systems served as inspiration or guidance for various aspects of LibFeature.

The idea of constructing a virtual matrix containing all of the results and then accessing only individual rows (feature vectors) as needed was inspired by the typical use of Structured Query Language (SQL) [1] within a high-level programming language. A program accessing an SQL database will typically use one function to execute an SQL query, which returns a handle to a result set (e.g., SQLExecute in the standard SQL Call Level Interface (SQL/CLI)). The program then calls a separate function to access individual rows from the result set (e.g., SQLFetch in SQL/CLI).

The idea of compiling a high-level language into an intermediate bytecode which can then be executed more quickly was inspired primarily by Java's bytecode [2] [3], which popularized the concept. Finally, the idea of using a somewhat restricted high-level language to specify instructions that are executed quickly on thousands of vectors in parallel is very similar to pixel shading languages such as the OpenGL Shading Language [4], which allows programmers to write simple programs that are executed directly on a graphics card to determine the final color of each pixel in a 3-D rendered image.

3 System Outline

LibFeature is initialized with one or more input arrays, and a Feature Description Language (FDL) program specifying how to create the feature vectors. On initialization, LibFeature parses the program but does not compute any actual feature vectors, but instead returns a *virtual matrix* of feature vectors. For example, if you have a 50×50 image of pixels and each feature vector has 7 features, the virtual matrix would be of size $2,500 \times 7$. (By convention, we assume that the matrix is stored in *row-major* order, in which case each feature vector is one row. It is equally correct, however, to imagine a column-major matrix where each feature vector is a column; this may be preferable to Fortran and Matlab users, and in either case the memory representation is the same.)

From a high-level language, you then query LibFeature for a single row of the virtual matrix (one feature vector), a set of contiguous rows, or an arbitrary list of rows. LibFeature is designed to work with very large virtual matrices that could never be computed and stored in memory all at once; specifically there is no limitation that the number of total elements in the matrix fit in one 32-bit integer. It is quite common to initialize a virtual matrix with ten trillion elements, but only actually compute a few thousand rows randomly scattered throughout the matrix.

Internally, LibFeature works with only floating-point numbers. While input arrays are allowed to be any numerical data type, everything is converted to (your choice of single- or double-precision) floating-point values first in the resulting feature vector. This was a reasonable simplifying design decision because on modern CPUs there is little performance difference between floating-point and integer calculations.

3.1 Input Arrays and Dimensions

LibFeature can read from arrays in memory, or directly from binary files on disk. When reading from disk, you can specify the data type, start offset, endianness, and dimensions, or LibFeature can determine it automatically from the file's header (supported file types currently include BMP, Matlab, NetCDF, PGM, PPM, and TIFF). (LibFeature can be used with numerical data stored in ASCII text files, too, but currently this requires reading the entire file into memory first, so it does not scale as well.) LibFeature works with arbitrary multidimensional arrays, but it requires that you specify which dimensions measure unique data points (extrinsic dimensions), and which dimensions measure different fields within the same data point (intrinsic dimensions). For example, consider a typical color image file with dimensions of 640 × 480 pixels, and three color components (red, green, and blue) per pixel. The total length of the file is 640 × 480 × 3 elements, however only 640 × 480 of these are unique data points. The last dimension is an intrinsic dimension, because it counts the number of elements within each pixel (in this case, 3). LibFeature uses the convention of using positive numbers to denote extrinsic dimensions, and negative numbers to denote intrinsic dimensions, given in the order of fastest-moving to slowest-moving.

Most color 640 × 480 images, then, would have dimensions $(-3, 640, 480)$ because the file is stored like this:

$$480\left\{\begin{array}{l} \overbrace{\text{RGB RGB RGB} \ldots \text{ RGB}}^{640} \\ \text{RGB RGB RGB} \ldots \text{ RGB} \\ \vdots \qquad\qquad\qquad \ddots \quad \vdots \\ \text{RGB RGB RGB} \ldots \text{ RGB} \end{array}\right.$$

Users of scientific image datasets may be used to calling this the Band-Interleaved-by-Pixel (BIP) format. However, some image files store the red, green, and blue channels as separate planes, meaning that the first 640 × 480 elements in the file are for the red channel, which is followed by all of the green channel, and then all of the blue. Scientific users would call this a Band-Sequential (BSQ) file. In this case, the dimensions that you pass to LibFeature would be $(640, 480, -3)$:

$$\begin{array}{l} 480\left\{\begin{array}{l} \overbrace{\text{R R R} \ldots \text{ R}}^{640} \\ \vdots \qquad\quad \ddots \ \vdots \\ \text{R R R} \ldots \text{ R} \end{array}\right. \\ 480\left\{\begin{array}{l} \text{G G G} \ldots \text{ G} \\ \vdots \qquad\quad \ddots \ \vdots \\ \text{G G G} \ldots \text{ G} \end{array}\right. \\ 480\left\{\begin{array}{l} \text{B B B} \ldots \text{ B} \\ \vdots \qquad\quad \ddots \ \vdots \\ \text{B B B} \ldots \text{ B} \end{array}\right. \end{array}$$

The default assumption in LibFeature is that you want to generate exactly one feature vector per data point. In section 4.1, we will see how to exclude some data points from getting turned into feature vectors, and in section 4.2, we will see how to generate multiple feature vectors from one data point. But most of the time there is a direct correspondence. The concept of extrinsic and intrinsic dimensions and LibFeature's convention of using negative indices is a surprisingly powerful way to represent mappings between the input array and feature vectors. As a more complicated example, suppose that you want to take the same image above, but you only want 320 × 240 feature vectors - one for each 2×2 group of pixels. All that is required is to tell LibFeature that the dimensions of the file are $(-3, -2, 320, -2, 240)$. LibFeature can infer from this that there will be 320×240 feature vectors, and makes it effortless for you to combine elements from the four pixels in each 2×2 group into each feature vector. Another more common illustration of the power of this convention is that when using LibFeature, switching from a BSQ to a BIP input file often requires changing only one line of code.

3.2 Feature Selection

A common task in machine learning problems is to start with a large feature set containing dozens or hundreds of potential features, and use a feature selection algorithm to choose a subset of features out of these. LibFeature is designed to make this quite easy and efficient. At any time, you can pass LibFeature a *mask array*, specifying which of the columns of the virtual matrix, corresponding to features, you are interested in. Not only will LibFeature take this into account, returning only the columns of the virtual matrix of interest from then on, but it also quickly re-optimizes its computation to avoid unneeded computations. Thus, there is no penalty for writing an FDL program to compute every possible feature you could imagine, and then later selecting the actual features you want to use empirically.

4 FDL: The Feature Description Language

The Feature Description Language (FDL) is used to specify how each feature vector is constructed from the input arrays. The syntax is based on Lisp-like S-expressions, and while it contains many capabilities commonly found in programming languages (including many built-in mathematical functions, conditionals, user-defined functions, and macros) it is an imperative language and purposefully does not include any flow control. The reason for the lack of flow control is that the same commands are applied to thousands of feature vectors in parallel, allowing for substantial optimizations.

Recall that S-expressions use *prefix* notation, where the name of the function or operator is always the first element of a parenthesized list, and the arguments follow. For example, the formula to compute the approximate area of a circle given its radius r could be expressed as:

```
(* 3.14159 (* r r))
```

An FDL program produces one feature vector given one particular input data point. The command to produce a feature is `output`. To retrieve the value of an input data point (pixel, in the case of an image), simply use the name of the input array (which was assigned during initialization) as if it was a function. Supposing that our RGB ($-3 \times 640 \times 480$) image was named `myimage`, then the simplest possible FDL program would be:

```
(output (myimage))
```

This program would only result in one feature per pixel. To output three features, one each for red, green, and blue, pass an argument to `myimage` indicating the offset of the element you want:

```
(output (myimage 0))
(output (myimage 1))
(output (myimage 2))
```

A 0 is always implied if there is no argument. Note that the offset can appear in any dimension, not just the first dimension. So it would be very easy to make each feature vector contain six features - the RGB values of the current pixel, and the RGB values of the pixel above and to the left of that one:

```
(output (myimage  0  0  0))
(output (myimage  1  0  0))
(output (myimage  2  0  0))
(output (myimage  0 -1 -1))
(output (myimage  1 -1 -1))
(output (myimage  2 -1 -1))
```

Note that the order of the offsets corresponds to the order of the dimensions: the first offset is relative to the fastest-moving dimension, and so on. Finally, note that when an offset takes you out of the bounds of an image, LibFeature replaces those features with NaN (not-a-number). It is easy to simply eliminate any feature vectors that have any NaNs in them later.

FDL lets you create variables using the `set` command, and it also comes with most standard mathematical functions. Here's a more complicated example, then, that outputs the three features per pixel, but normalizes them so that they always sum to 1:

```
(set red   (myimage 0))
(set green (myimage 1))
(set blue  (myimage 2))
(set sum (+ red green blue))
(output (/ red sum))
(output (/ green sum))
(output (/ blue sum))
```

This works as expected, unless the red, green, and blue values happen to all be 0. LibFeature will properly return NaNs when this happens, but if you would prefer to output zeros instead, use a conditional: `(if` *expr true-value false-value*`)`

```
(set red   (myimage 0))
(set green (myimage 1))
(set blue  (myimage 2))
(set sum (+ red green blue))
(output (if (== sum 0) 0 (/ red sum)))
(output (if (== sum 0) 0 (/ green sum)))
(output (if (== sum 0) 0 (/ blue sum)))
```

Note that like C, Java, Perl, etc., Libfeature uses a double-equals (`==`) to test for equality.

As a final example, here is an FDL program that averages the pixel values over a 3×3 region. This will introduce a `for` loop. Even though FDL does not have any loops that execute a different number of times depending on the status of real-time calculations, it does include constructs that allow loops that execute a fixed number of times. There are more variations, but the simplest syntax for a `for` loop is `(for` *variable* `(seq` *start stop*`)` *command* `[`*commands...*`])`, where `seq` is a built-in function that returns a list of elements to be iterated over, from *start* to *stop*, inclusive. Here's the code:

```
(set red 0)
(set green 0)
(set blue 0)
(for i (seq -1 1)
  (for j (seq -1 1)
    (set red (+ red (myimage 0 i j)))
    (set green (+ green (myimage 1 i j)))
    (set blue (+ blue (myimage 2 i j)))))
(output (/ red 9))
(output (/ green 9))
(output (/ blue 9))
```

4.1 Validation

In the most recent example FDL program above, one problem is that for all of the pixels along the edges of the image, the 3×3 neighborhood extends outside the image bounds, resulting in NaNs. One way to solve this would be to change the FDL code so that it only sums the values if they are not NaN. However, sometimes it might make more sense to simply exclude those pixels from becoming feature vectors. So for our 640×480 image, we would only end up with 638×478 feature vectors. In LibFeature this can be done using the `require` command. When the argument to `require` evaluates to false, the entire feature vector is marked as invalid. (Then at runtime, the user can choose to extract all feature vectors in a given range, or only the valid feature vectors in that range.) To test if a value is not NaN, you can use the `finite` function. Here is the above program with validation:

```
(set red 0)
(set green 0)
(set blue 0)
(for i (seq -1 1)
  (for j (seq -1 1)
    (set red (+ red (myimage 0 i j)))
    (set green (+ green (myimage 1 i j)))
    (set blue (+ blue (myimage 2 i j)))))
(require (finite red))
(require (finite green))
(require (finite blue))
(output (/ red 9))
(output (/ green 9))
(output (/ blue 9))
```

Note that it would have worked equally well to put the `require` inside of the for loop. When there are a lot of feature vectors that will be marked as invalid, it is usually fastest to do the validation as early as possible, because LibFeature can stop computing the feature vector as soon as it is marked invalid. When most vectors will be valid, it's best to execute the `require` command as few times as possible.

4.2 Variations

One way to cut down on overfitting in machine learning problems is to *jitter* the input, turning each input point into multiple feature vectors, each one offset by a small amount. LibFeature makes it easy to have one input point result in multiple feature vectors. Simply pass an initial argument to the `output` and `require` commands indicating the 0-based index of the feature vector to be output. Each different feature vector is then called one *variation.* Here's a simple FDL program that outputs two feature vectors per pixel: one vector is the original pixel, and the other is a darker version of the same pixel:

```
(set red   (myimage 0))
(set green (myimage 1))
(set blue  (myimage 2))
(output 0 red)
(output 0 green)
(output 0 blue)
(output 1 (* 0.9 red))
(output 1 (* 0.9 green))
(output 1 (* 0.9 blue))
```

4.3 Multiple input arrays

In all of the examples above, we have been assuming that LibFeature took just a single input array. However, LibFeature was designed to work with multiple inputs. There are two common ways that multiple inputs are used. The first way is when the data for a single image or other dataset is already stored in multiple files. For example, a color image might be stored as three separate files named `myimage.red`, `myimage.green`, and `myimage.blue`. In this case, you would initialize LibFeature with all three arrays and give them different names. Since the dimensions of all of the files are the same, LibFeature would automatically align them and generate only one set of feature vectors. The second way that multiple inputs can be used is when you want to concatenate feature vectors from multiple, independent files. If you initialize LibFeature with multiple input files but give them all the same name, LibFeature will process feature vectors from the files consecutively. So if you had two images, one 640×480 and one 800×600, LibFeature would return a virtual matrix with $640 \cdot 480 + 800 \cdot 600 = 787,200$ rows. This is particularly useful for training machine learning algorithms on a random subset of vectors. Rather than computing feature vectors for every pixel in hundreds of images, simply initialize LibFeature with all of the images, get one virtual matrix, and then retrieve random rows from that matrix until training has converged.

4.4 Labels

Training a supervised classifier requires a class label for every feature vector. While it would be possible to simply denote the last element of each feature vector as the class label, in practice it is often convenient to have the labels available separately. LibFeature provides a special mechanism to output a single label associated with each vector, using the `label_output` command.

5 Limitations and workarounds

While FDL is powerful enough to do many computations and generate surprisingly complex feature vectors (see section 7.2), there are certainly several types of features that FDL is not adept at expressing. In particular, FDL assumes that each feature vector can be generated independently and deterministically. As a simple example, there is no way in FDL to specify that a particular feature is to be normalized, because this would require scanning through the entire input first. One solution is to use LibFeature in two passes: In the first pass, LibFeature can be used to scan the input files and return vectors containing the values to be normalized. The program can then compute the normalization coefficients, and call LibFeature a second time with these coefficients in the FDL. As another example, it is common to compute the wavelet decomposition of an image to derive texture features, but FDL is not designed for transformations that operate on an entire image at once. In this case, it is best to use another program to preprocess input images and generate transformed images. LibFeature can still be useful for generating feature vectors from these transformed inputs, though, to combine the untransformed and transformed inputs with different weights.

6 Inside LibFeature

In this section, we will examine how FDL programs are converted to an internal representation and executed to produce the feature vectors. Readers who are only interested in using LibFeature as a black box may want to skip directly to section 7 or 8. So as to illustrate the diversity of problems for which LibFeature is applicable, we will switch from using an image data set as the main example to a time-series analysis problem. Suppose that your data contains several million readings taken at periodic intervals from some scientific instrument. To make things more interesting, also suppose that the data is somewhat noisy, and you want to smooth it out using a very simple convolution filter: every point will be replaced with the mean of the values within a certain neighborhood centered at that point. Here's an FDL program that computes a feature vector consisting of each value and the mean of its temporal neighborhood, supposing that the input file is named `sensor_input`:

```
(set nborhood 2)
(set len (+ 1 (* 2 nborhood)))
(for i (seq (- 0 nborhood) nborhood)
  (append a (sensor_input i)))
(set psum (+ a))
(output 0 (sensor_input 0))
(output 0 (/ psum len))
```

When LibFeature parses this FDL program, it interprets each S-expression in order, building up its internal data structures. Every expression it encounters turns into a command, where some commands use references to previous commands as parameters. Note that `set` does not result in a command, but instead adds an entry to a symbol table, allowing you to refer to the result of a particular expression by name. Here is a representation of the commands that are generated when the FDL program above is parsed:

```
v[0] = 2;
v[1] = 5;
v[2] = array("sensor_input", -2);
v[3] = array("sensor_input", -1);
v[4] = array("sensor_input", 0);
v[5] = array("sensor_input", 1);
v[6] = array("sensor_input", 2);
v[7] = sum(v[2], v[3], v[4], v[5], v[6]);
v[8] = array("sensor_input", 0);
v[9] = v[7] / v[1];
```

This representation is meant to give you an idea of how the particular feature vector described above could be computed using a temporary array `v[]`. The array `v[]` is analogous to register on a processor. Internally, LibFeature represents these commands using a bytecode, but the information contained is the same. In this particular case, after the nine commands are finished for each input point, the outputs are in `v[8]` and `v[9]`.

6.1 Optimizations

Note that there is a little bit of redundancy in the sequence of commands shown above. The first value, `v[0]`=2 (which came from `nborhood`) is never actually used. Also, `v[4]` and `v[8]` are identical. LibFeature performs some optimizations on its command sequence before executing it. These optimizations are very similar to optimizations performed by modern compilers, but because there is no flow control, they are all relatively straightforward, and in fact all of the optimizations run in time $O(n \log n)$ (where n is the number of commands). The optimizations performed by LibFeature include:

1. Constant Propagation: Expressions that return constant values are replaced with the result of that expression. This was already seen in the above command sequence, because the expression `(+ 1 (* 2 nborhood))` was replaced with the result, 5, in a single step.
2. Identity Reduction: Computations that involve multiplying by one or adding zero are eliminated. This encourages users to write code with lots of tunable parameters, since an additive parameter can simply be set to 0 to eliminate it.
3. Common Subexpression Elimination: Commands that are duplicates of earlier commands are eliminated.
4. Dead Code Elimination: Commands which are never used are eliminated.

After optimization, the command sequence looks like this:

```
v[0] = 5;
v[1] = array("sensor_input", -2);
v[2] = array("sensor_input", -1);
v[3] = array("sensor_input");
v[4] = array("sensor_input", 1);
v[5] = array("sensor_input", 2);
v[6] = sum(v[1], v[2], v[3], v[4], v[5]);
v[7] = v[6] / v[0];
```

The outputs are now in `v[3]` and `v[7]`.

6.2 Execution

In the simplest case, when LibFeature is computing just a single feature vector, the execution model is quite simple: it executes the commands in the bytecode on a temporary array (`v[]` in the example above), and then copies the lines from the array corresponding to actual features into the feature vector at the end. LibFeature is much more efficient, however, when it is given the opportunity to compute multiple feature vectors in parallel. Every command operates on many feature vectors simultaneously. This vectorization allows the overhead of interpreting the program to be minimized, and LibFeature can start approaching the speed of C code.

For example, suppose that the first few numbers in our `sensor_input` data happen to be the digits of π: $[3, 1, 4, 1, 5, 9]$. In executing the commands to generate the first four feature vectors, LibFeature would fill in a matrix with the following values:

	Feature Vectors			
Command	1	2	3	4
v[0] = 5;	5	5	5	5
v[1] = array(-2);	NaN	NaN	3	1
v[2] = array(-1);	NaN	3	1	4
v[3] = array(0);	3	1	4	1
v[4] = array(1);	1	4	1	5
v[5] = array(2);	4	1	5	9
v[6] = sum(v[1], ..., v[5]);	NaN	NaN	14	20
v[7] = v[6] / v[0];	NaN	NaN	2.8	4.0

Note that since LibFeature cannot index negative elements in the input array, it replaces these values with NaN. After all of the computations are finished, LibFeature still needs to copy the rows corresponding to output features to the completed feature vectors in memory. When multiple feature vectors are computed at once, individual features of the same feature vector are never contiguous, so the extra copy is always required. While this does take some extra time, it is negligible compared to the time that is saved by being able to compute each of the rows of the temporary matrix all at once, making use of the fact that the elements of each row are consecutive in memory. For example, the inner loop of the code which computes the division command in the last step is actually something like this:

```
for(i=0; i<len; i++) {
   *out = *in1 / *in2;
   out++;
   in1++;
   in2++;
}
```

In practice, as many as 64 or 128 feature vectors are usually computed at once, and modern C compilers are able to make this loop extremely efficient. By vectorizing all of these computations (always doing one step of many feature vectors at once), we can minimize the overhead of LibFeature. Furthermore, LibFeature takes advantage of vector instructions such as SSE on x86 chips, or AltiVec on PowerPC chips, to make these inner loops run even faster. Using these instructions for many common arithmetic operations speeds up LibFeature by 25% overall in typical usage.

6.3 Multithreading

Since many scientific workstations have dual processors (and other modern CPUs simulate multiple processors using hyperthreading), LibFeature is designed to transparently take advantage of these processors by using two threads for computation.

In cases where all of the input data is to be loaded, using two threads is trivial: one thread loads the first half of the data, and the other thread loads the second half. However, LibFeature allows for validation of individual feature vectors (using the `require` statement in FDL), which complicates matters significantly. Since it is unknown exactly how many feature vectors will be generated, the two threads must communicate frequently in order to write their feature vectors to the same array without leaving any gaps (which would require costly memory moving later, negating the benefit of both threads). Pipes are used both for thread synchronization and communication. The speedup when using two threads on a dual-processor machine is typically 1.5x, though it can be as high as 1.9x for some computationally intensive programs.

7 Benchmarks

LibFeature has been designed to provide no performance penalty in spite of the fact that it interprets its commands. The following benchmarks demonstrate how closely this goal has been achieved.

7.1 Image features

Consider a simple problem where we wish to extract two features from a grayscale 1600×1200 PGM image: the intensity of each pixel, and the statistical variance of an $n \times n$ neighborhood of each pixel. To keep the neighborhood symmetrical, suppose $n = 2 \cdot radius + 1$ for some $radius \geq 0$. Here is an example of straightforward C code to implement this feature extraction:

```
void get_features(FILE *fp,
                  float *features, /* output */
                  int radius,
                  int width, int height)
{
   uint8 *input = (uint8 *)malloc(width * height);
   float *p = features;
   int i, j, k;
   int len = (1+(radius*2))*(1+(radius*2));

   fread(input, 1, width * height, fp);

   for(i=0; i<width * height; i++) {
      float sum = 0;
      float sumsq = 0;
      float var;

      for(j=-radius; j<=radius; j++)
         for(k=-radius; k<=radius; k++) {
            int index = i + j + (k*width);
            if (index >= 0 &&
                index < width * height) {
               float v = (float)input[index];
               sum += v;
               sumsq += v * v;
            }
         }

      var = (sumsq - ((sum * sum) / len)) / len;

      *p++ = (float)input[i];
      *p++ = var;
   }
}
```

Now here is the same code, implemented in FDL:

```
(set width (+ 1 (* 2 radius)))
(set len (* width width))
(for i (seq (- 0 radius) radius)
  (for j (seq (- 0 radius) radius)
    (set v (image i j))
    (append array1 v)
    (append array2 (* v v))))
(set sum1 (+ array1))
(set sum2 (+ array2))
(set center (image))
(set var1 (- sum2 (/ (* sum1 sum1) len)))
(set var (/ var1 len))
(output center)
(output var)
```

It requires at most five lines of C code to extract feature vectors from an image given the FDL program above. The LibFeature solution is more compact than the C code, and significantly more

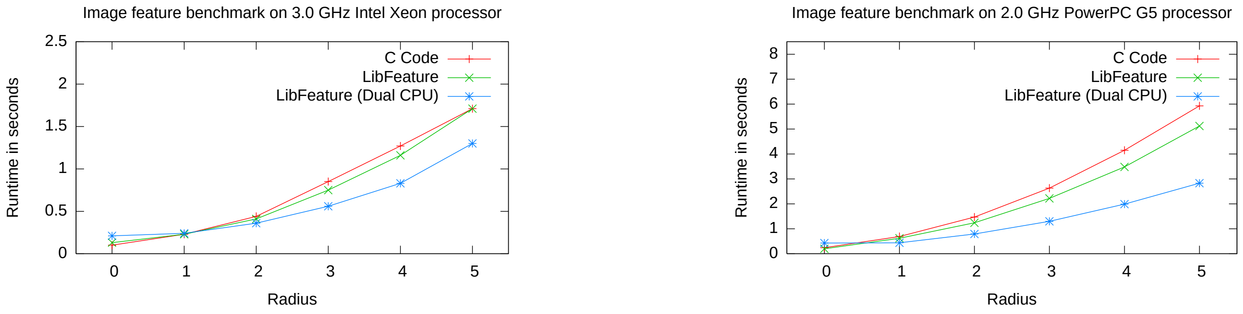


Fig. 1. Benchmark results show that for this particular problem, LibFeature is comparable in speed to straightforward C code when a single processor is used, and somewhat faster when two processors are used.

flexible. To show that there is also no performance penalty, we ran the C code against LibFeature on two different scientific workstations, for several different radii. The following times are measured in seconds:

	Xeon/3.0 GHz		G5/2.0 GHz	
Radius	C	LibFeature	C	LibFeature
0	0.10	0.13	0.24	0.20
1	0.23	0.23	0.69	0.62
2	0.44	0.41	1.47	1.24
3	0.85	0.75	2.63	2.22
4	1.27	1.16	4.15	3.48
5	1.71	1.71	5.93	5.12

Many modern scientific workstations have dual processors. While it is uncommon for a programmer to write multithreaded code for something as simple as feature extraction, LibFeature can take advantage of multiple processors safely and without any extra work on the part of the programmer. While this can introduce extra overhead for small easy problems, in most cases this allows LibFeature to run significantly faster than ordinary C code:

	Dual Xeon/3.0 GHz		Dual G5/2.0 GHz	
Radius	C	LibFeature	C	LibFeature
0	0.10	0.21	0.24	0.43
1	0.23	0.24	0.69	0.44
2	0.44	0.36	1.47	0.79
3	0.85	0.56	2.63	1.30
4	1.27	0.83	4.15	1.99
5	1.71	1.30	5.93	2.83

7.2 FFT

While FDL is not a complete programming language (it does not have any flow control), it is nevertheless powerful enough to compute many complicated algorithms. As an example of this, consider the Fast Fourier Transform (FFT) [5], commonly used to extract frequency features from time-series data. Suppose that you have a million time points, and for every point you wish to create a feature vector containing the Fourier Transform of the n points centered at that point. For simplicity assume that n is a power of two and that we are computing the complex FFT. We implemented a general power-of-two complex FFT in only 46 lines of FDL code, using a very straightforward nonrecursive algorithm. We benchmarked it against FFTW 3.0.1 [6] [7], widely regarded as the one of the fastest FFT implementations available for any computing platform. We tested them both in single-precision mode without making use of multiple processors; the times below are in microseconds, per FFT, when computing at least 100,000 FFTs in a row.

	Xeon/3.0 GHz		G5/2.0 GHz	
FFT Size	FFTW	LibFeature	FFTW	LibFeature
16	0.54	0.94	0.55	0.66
32	1.19	2.00	1.00	1.46
64	2.11	4.63	1.85	3.43
128	4.71	12.15	4.54	8.43

7.3 Analysis of benchmark results

While it is not necessarily unexpected that LibFeature can be more efficient than straightforward C code when using two processors, this does not explain how it matches and sometimes beats the performance of C code even when using a single processor, as seen above. Contributing to LibFeature's efficiency:

1. Cache efficiency: The C code in section 7.1 skips through memory in order to retrieve the values for each feature vector, resulting in inefficient use of the processor's L1 cache. LibFeature typically copies 64 contiguous values at a time from the input array, directly to a row in the temporary matrix.
2. Vector instructions: On both x86 and PowerPC processors, LibFeature uses vectorized addition, multiplication, and division, which allows it to operate on four values at once.
3. Memory mapping: Instead of loading the file into memory in a single `fread` command (which can take some time to complete), LibFeature memory-maps the file, which allows the computation to begin on the first few bytes of the file while the rest is still being loaded.

It is not surprising that FFTW is faster than LibFeature; it has been extensively tuned and undergone several major revisions and has essentially no overhead. The fact that LibFeature is only a factor of 2-3 slower than FFTW, given only 46 lines of high-level FDL code, is impressive and serves as validation of LibFeature's design and architecture. Note of course that LibFeature has more overhead and must be given thousands of FFTs to compute in order to approach this speed. Also, to be fair, LibFeature consumes vastly more memory than FFTW needs to for the same task.

8 Conclusions and Future Work

LibFeature is designed to make life easier for the machine learning or data mining researcher. Instead of wasting time writing complicated transformations to construct feature vectors from input data, you can use LibFeature to do this work for you. By abstracting the feature vector generation from the algorithm, it becomes easier to change the feature vectors on the fly; experimenting with new ideas for features thus requires less effort. Because LibFeature was implemented very carefully with performance in mind, it is usually possible to use LibFeature and pay no performance penalty at all. In fact, in many circumstances, LibFeature is significantly faster than straightforward alternatives.

We are making use of LibFeature in several projects, and many new capabilities are constantly being added. Some possible new capabilities that we are considering for the future include better support for reading numerical data from ASCII text files, more binary data formats, more language bindings, dual-pass algorithms for normalized features, integer features, and efficient random shuffling of vectors.

9 Acknowledgements

Thanks to Dennis DeCoste for the initial idea and supporting this work, and to Ben Bornstein for helpful suggestions on this paper. The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

10 Software Availability

Information on obtaining LibFeature is available from the JPL Machine Learning Group web page: `http://ml.jpl.nasa.gov/`

References

1. Date, C.J., Darwen., H.: A Guide to the SQL Standard, 4th Edition. Addison-Wesley (1997)
2. Gosling, J., Joy, B., Steele, G.: The Java Language Specification. Addison Wesley (1997)
3. Gosling, J.: Java intermediate bytecode. In: Proceedings of the Workshop on Intermediate Representations ACM SIGPLAN Notices. Volume 30. (1995) 111 – 118
4. Post, R.J.: OpenGL Shading Language. Addison-Wesley (2004)
5. Cooley, J., Tukey, J.: An algorithm for the machine computation of the complex fourier series. Mathematics of Computation **19** (1965) 297–301
6. Frigo, M., Johnson, S.G.: FFTW: An adaptive software architecture for the FFT. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. Volume 3., Seattle, WA (1998) 1381–1384
7. Frigo, M., Johnson, S.G.: FFTW 3.0.1 (software) (2003) `http://www.fftw.org/`.

A Data Mining Approach to Matrix Preconditioning Problem

Shuting Xu, [*] Jun Zhang [†]
Laboratory for High Performance Scientific Computing and Computer Simulation,
Department of Computer Science, University of Kentucky,
Lexington, KY 40506–0046, USA

February 22, 2005

Abstract

Preconditioned Krylov subspace methods are generally used to solve large sparse linear systems with sparse coefficient matrices. However, choosing an efficient preconditioner for a specific sparse matrix arising from a particular application presents a formidable challenge for many design engineers and application scientists who have little knowledge of preconditioned iterative methods. We propose to build an Online Preconditioner Prediction System, which can predict the solvability of general sparse matrices with structure-based sparse matrix preconditioners using data mining techniques. We first group matrices into clusters, then use Support Vector Machine Classification technique to predict the solving status of the sparse matrices in clusters with low purity values. The prediction method can predict whether a matrix can be solved by a preconditioner (in a preconditioned iterative solver). In the case of a negative prediction, the method can also predict the possible reasons why the matrix cannot be solved by this preconditioner. Our experimental results show that the overall accuracy of the prediction is above 90% for the ILU0 preconditioner and above 87% for the ILUK preconditioners.

Key words: sparse matrix, support vector machine, preconditioning

1 Introduction

There are many scientific applications in which there is a need to solve large sparse linear systems with sparse coefficient matrices, e.g., the applications to develop the next generation electromagnetics simulators [12], to track nerve fibers in human brains [10], and to simulate laminar diffusion flames [11], to name just a few. The class of preconditioned Krylov subspace methods [19] (with a Krylov iterative solver and a preconditioner) is considered the preferred methods in this field. The preconditioners employed in the preconditioned iterative solvers usually determine the overall convergence rate of the iteration procedure [25]. However, choosing a good preconditioner for a specific sparse matrix arising from a particular application is the combination of art and science, and presents a formidable challenge for many design engineers and application scientists who have little knowledge of preconditioned iterative methods.

There is an enlarging gap between the development of more and more sophisticated preconditioned iterative solvers by the computational linear algebra community and the ability to understand and to properly use these solvers by the application scientists and engineers to solve their more and more complex modeling and simulation problems. High performance computers and numerical algorithms will be less useful if they are not matched with the intended application problems.

Current approaches to recommending iterative solvers and preconditioners are quantitative and categorical. Although mathematical theorems can be proved to guarantee the convergence of certain preconditioned iterative methods for a given model problem, they are not very useful for solving problems encountered in practical applications. Thus a recommendation approach that permits tradeoffs and that can be built and modified incrementally, based on increased knowledge, is certainly useful. The success of the software environment approach for other problems in scientific computing [6] suggests that it can be very useful when a recommendation system or a software environment approach is applied to the problem of finding the right path to solving a sparse matrix.

There has been a considerable amount of effort made by several researchers and organizations to collect various sparse matrices in order to use them for

---

[*] The research work of S. Xu was supported by NSF under grant ACR-0234270. E-mail: sxu2@uky.edu, URL: http://www.csr.uky.edu/∼sxu2.

[†] Correspondent. The research work of J. Zhang was supported in part by NSF under grants CCR-0092532 and ACR-0202934, by DOE under grant DE-FG02-02ER45961, and by the University of Kentucky Research Committee. E-mail: jzhang@cs.uky.edu, URL: http://www.cs.uky.edu/∼jzhang.

test purposes. The National Institute of Standard and Technology (NIST) has been playing a leading role in this endeavor and currently hosts one of the largest such repositories: MatrixMarket [15]. Several other collections have been contributed by engineers, scientists and numerical analysts, e.g., the well-known Harwell-Boeing sparse matrix collection and the University of Florida sparse matrix collections [4]. NIST has done some categorization work and published some preliminary information on these matrices. For each matrix this information includes its type, dimensions, condition number, nonzero structure, etc. MatrixMarket is becoming a standard source of sparse matrices for testing various direct and iterative solution methods. However, there is no information regarding which matrix can be solved by what method using what parameters. Such information would be extremely helpful for application scientists and engineers as it would enable them to choose suitable sparse matrix solvers for certain class of applications. Furthermore, solving some of the matrices in the MatrixMarket does not provide much information to help application scientists and engineers whose matrices in question may not match any of the matrices in the MatrixMarket exactly.

The first idea of using matrix features and data mining techniques to predict the possibility of solving a sparse matrix was proposed in [26]. We are now working on a project to build an online Intelligent Preconditioner Recommendation System (IPRS) [24, 23], which can provide expert advice on choosing a high performance preconditioner as well as the suitable parameters for a given sparse matrix. The first step of the project is to design and implement an Online Preconditioner Prediction System (OPPS), which is the main topic of this paper. That is, given a sparse matrix, predicting whether the matrix can be solved by a certain preconditioned solver (preconditioner). If not, the system will try to predict the reason why it fails.

The OPPS system works as follows. First, we extract features from matrices, then these features are used to classify the matrices into groups, with matrices in the same group having the same solving status. There are many classification algorithms in literature. We choose Support Vector Machine Classification (SVC) [20, 21] because of its success in many classification problems. We also propose a new prediction method. The matrices are first divided into clusters according to their similarity in the features. We define the *purity* of a cluster to be the maximum percentage of the sparse matrices in the cluster that have the same solving status. Then SVC is applied only to the clusters with low purity values. Our experimental results show that we not only improve the accuracy of the prediction by the clustered SVC method but also gain some insights into the relationship between matrix features and solvability of the matrix by the preconditioned solvers.

We introduce preconditioners in Section 2 of this paper. In Section 3, we describe the structure of the Online Preconditioner Prediction System. Some of the identified matrix features are explained in Section 4. We briefly review the clustering and classification algorithms used in the system in Section 5. The proposed prediction method is given in Section 6. The computational experiments are carried out and the results are discussed in Section 7. We sum up this paper in Section 8.

2 Preconditioners

The most promising general purpose iterative solution methods for solving large sparse matrices is the class of preconditioned Krylov methods [19]. Preconditioners in an explicit form consist of an explicitly constructed matrix M, which is close to A in some sense. The key criteria for a good explicit preconditioner M are that the construction cost of M as well as the solution cost of an auxiliary linear system with M should be inexpensive.

The Generalized Minimal Residual Method (GMRES) is a projection method based on the Krylov subspace. We apply the PGMRES [19] solver which uses the L and U matrices generated from the respective preconditioner $M = LU$ to precondition the original matrix before the application of the GMRES algorithm. The preconditioner is applied to the right hand side. The preconditioners that we choose to study in this paper are ILU0 and ILUK.

ILU0 is the incomplete LU (ILU) factorization technique with no fill-in, which takes the zero pattern P to be precisely the zero pattern of the matrix A [14, 19]. The definition of the ILU0 factorization is: "any pair of the matrices L (unit lower triangular) and U (upper triangular) so that the elements of $A - LU$ are zero in the locations of $NZ(A)$", where $NZ(A)$ is the nonzero pattern of the matrix A. ILU0 is simple to implement, and its computational cost is inexpensive. It is effective for some problems, such as those from low-order discretizations of scalar elliptic PDEs and diagonally dominant matrices. However, for more difficult and realistic problems the no fill-in ILU0 factorization results in too crude an approximation to the original matrix, and more sophisticated preconditioners, which allow some fill-in elements in the incomplete LU factors, are needed [1].

ILUK is the ILU factorization with level of fill to be k. The initial level of fill of an element a_{ij} of a sparse matrix A is defined by:

$$\mathrm{lev}_{ij} = \begin{cases} 0, & \text{if } a_{ij} \neq 0, \text{ or } i = j \\ \infty, & \text{otherwise.} \end{cases}$$

Each time the value of a_{ij} is modified in the Gaussian Elimination procedure in the preconditioner construction, its level of fill is updated by:

$$\text{lev}_{ij} = \min\{\text{lev}_{ij}, \text{lev}_{ik} + \text{lev}_{kj} + 1\}.$$

In ILUK, all fill-in elements whose level of fill does not exceed k are kept. In this paper, we consider only $k = 1, 2, 3$ respectively. There are also some drawbacks of ILUK. For example, the amount of fill-in elements cannot be predicted in advance and the cost of updating the levels can be quite high. Furthermore, the algorithms may drop large size elements and result in an inaccurate incomplete factorization for certain indefinite matrices [18].

We choose ILU0 and ILUK because they are relatively simple, and they do not have many free parameters used in the construction of the preconditioners that can influence their performance. Thus the results of such preconditioned iterative solvers are easier to predict. We will deal with more sophisticated preconditioners in our later work.

3 Online Preconditioner Prediction System

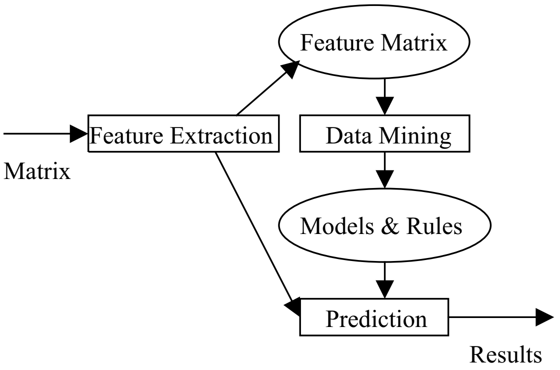


Figure 1: Online Preconditioner Prediction System.

The purpose of the Online Preconditioner Prediction System (OPPS) is to predict whether a sparse matrix can be solved by a preconditioned solver accurately and with a quick response time. The main structure of the OPPS is described in Figure 1. After a matrix is submitted, its features are computed and saved in a feature matrix. Some data mining algorithms are applied on the feature matrix to find out models, rules or other kinds of knowledge about the matrix features and their relationship with the preconditioned solvers. Such models, rules or knowledge are used to predict the solving status of the matrix, such as whether the sparse matrix can be successfully solved by a particular preconditioner, if not, which kind of problems will occur, etc. The procedures like mining the feature matrix are done in background and are repeated after some new matrices are submitted to the system. With more and more matrices added to the system, the prediction accuracy will improve. The response time of the system is just the time used to extract the matrix features and to predict the solving status based on the known knowledge.

4 Matrix Feature Extraction

The features of a matrix used are directly related to the precision of the prediction system. We will compute the features of a matrix first and then use such information to predict its solving status. The features of a matrix are a reflection of its sparsity and the locations and the size of the nonzero elements. We have extracted about 66 features such as the matrix structure, value, bandwidth and diagonal related statistics. There may be other useful features that we can extract in the future and add them to the feature space.

4.1 Structure This group of features describe the distribution of nonzero entries of a matrix. For example, we compute the sparsity rate (the number of nonzero elements divided by the number of all elements) of the whole matrix ($nnzrt$), of the lower diagonal part ($lowfillrt$), of the upper diagonal part ($upfillrt$) and of the main diagonal ($diagfillrt$). Other features include the average nonzero entries per row ($avnnzprow$) and the standard deviation ($sdavnnzprow$), the average nonzero entries per column ($avnnzpcol$) and the standard deviation ($sdavnnzpcol$), the maximum and minimum number of nonzero elements per column and per row ($maxnnzpcol, minnnzpcol, maxnnzprow, minnnzprow$). Sometimes we also want to know the total number of non-void diagonals ($nzdiags$), i.e., the number of diagonals which have at least one nonzero element among the $2n - 1$ diagonals of the matrix.

The attribute $symmc$ measures whether a matrix is symmetric, i.e., $A = A^T$. $relsymm$ describes the relative symmetry rate of a matrix. It is the ratio of the number of elements that match divided by nnz. An element $a(i, j)$ in the matrix A matches if it satisfies the following condition: if $a(i, j)$ is nonzero then $a(j, i)$ is nonzero. If a matrix is a normal matrix, $normal$ is equal to 1, otherwise it is 0.

The attribute $blocksize$ reflects whether a matrix has a block structure or not. The matrix has a block structure if it consists of square blocks that are dense. The value of $blocksize$ greater than one represents the size of the largest block.

4.2 Value The attributes in this group sum up the value distribution of a matrix, e.g., the one norm (*onenorm*), infinity norm (*infnorm*), and the Frobenius norm (*frnorm*) of a matrix. This group also includes the minimum of the sum of the columns (*minonenorm*), the minimum of the sum of the rows (*mininfnorm*), the Frobenius norm of the symmetric part of a matrix (*symfnorm*), and of the unsymmetric part (*nsymfnorm*).

We also consider average value of all nonzero entries (*avnnzval*) and the standard deviation (*sdavnnzval*), the average of the main diagonal entries (*avdiag*) and the standard deviation (*sdavdiag*), the average of the upper triangular entries (*avuptrig*) and the standard deviation (*sdavuptrig*), as well as the average of the lower triangular entries (*avlowtrig*) and the standard deviation (*sdavlowtrig*).

4.3 Bandwidth This group of features describe the bandwidth of a matrix. Bandwidth provides a measure of the clustering of nonzero entries about the main diagonal. Lower bandwidth of a matrix (*lowband*) is defined as the largest value of $i - j$, where $a(i, j)$ is nonzero. On the contrary, upper bandwidth of a matrix (*upband*) is defined as the largest value of $j - i$. Maximum bandwidth (*maxband*) is defined as $\max(\max(j) - \min(j))$. Average bandwidth (*avband*) is defined as the average width of all columns.

4.4 Diagonal The features in this group are diagonal related. For instance, they include the average distance from each entry to the diagonal (*avdisfd*) and the standard deviation (*sdavdisfd*), the average of the difference from each of the entry to its diagonal value (*avvalfd*) and the standard deviation (*sdavvalfd*), the average of the difference from the largest value in a row to the diagonal value (*avmaxvalfd*) and the standard deviation (*sdavmaxvalfd*).

Other features in this category include the percentage of weakly diagonally dominant columns (*diagdomcol*) and the percentage of weakly diagonally dominant rows (*diagdomrow*). *diagvalrate* is the ratio of the minimum absolute diagonal element value (except zero) to the maximum absolute diagonal element value.

4.5 Others We also include some other features, such as *strzpiv* - the number of structural zero pivots, (a structural zero pivot is a null column above or null row to the left of a zero diagonal element); *zpivrow* - whether a matrix has a null row to the left of a zero diagonal element; *zpivcol* - whether a matrix has a null column above a zero diagonal element; *szvdiag* - the smallest nonzero diagonal element with the dot product of its left vector and up vector being zero; *minvalcol* - the minimum of the smallest nonzero value in each column with a zero diagonal element.

For more detailed description on the identified matrix features, please see [23].

5 Clustering and classification

In this section, we briefly review the clustering and classification algorithms we use in the system.

5.1 K-means clustering The K-means algorithm [13] (with its many variants) is a popular clustering method in data mining. It gains its popularity due to its simplicity and intuition. The algorithm is an iteration procedure and requires that the number of clusters, k, be given *a priori*. Suppose that the k initial cluster centers are given, the algorithm iterates as follows:

(1) It computes the Euclidean distance from each of the objects to each cluster center. An object is assigned to the cluster with the smallest distance.

(2) Each cluster center is recomputed to be the mean of its constituent objects.

(3) Repeat steps (1) and (2) until the convergence is reached.

The criterion function for the convergence can be computed, e.g., as

$$f_r = \frac{1}{n} \sum_{i=1}^{n} (\mathrm{Edist}^2(d_i, c_j^{(r)})),$$

where r is the step of the iterations. The function $\mathrm{Edist}(d_i, c_j)$ computes the Euclidean distance between the object d_i and a cluster center c_j. Given a convergence criterion ϵ, the K-means algorithm stops when $|f_{r+1} - f_r| < \epsilon$. Note that f_r is a monotonically decreasing function with a lower bound. So its limit exists [5].

5.2 SVM classification The SVM (Support Vector Machine) is based on a structural risk minimization theory [20]. It has been successfully applied to many applications like face identification, text categorization, bioinformatics, etc [2, 3, 9].

In SVM classification, the goal is to find a hyperplane that separates the examples with maximum margin. Given l examples (x_1, y_1), ..., (x_l, y_l), with $x_i \in R^n$ and $y_i \in \{-1, 1\}$ for all i, SVM classification can be stated as a quadratic programming problem:

$$\text{minimize } \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{l} \xi_i$$

$$\text{subject to} \begin{cases} y_i(< w, x_i > +b) \leq 1 - \xi_i \\ \xi_i \geq 0 \\ C > 0 \end{cases}$$

where C is a user-selected regularization parameter, and ξ_i is a slack variable accounting for errors. After solving it, we can get the following decision function:

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i < x_i, x > +b. \tag{5.1}$$

where $0 \leq \alpha_i \leq C$.

For the nonlinear case, we apply a mapping $\Phi : X \to F$ to map the input space into some feature space F. Here we use a kernel function, $K(x, x_i) =< \Phi(x), \Phi(x_i) >$, which is a symmetric function and satisfies the Mercer's condition. We substitute $K(x, x_i)$ for the dot product, which maps the input space into some reproduced kernel feature space. Then Equation (5.1) can be rewritten as:

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i K(x, x_i) + b. \tag{5.2}$$

The above methods only work for classifying two classes. If the matrix features are to be classified into more than two classes, we have to use one of the multi-class classification methods [7, 16, 22]. Two of the commonly used multi-class methods are "one-against-one" and "one-against-all". Suppose there are n classes, "one-against-all" method constructs n classifiers. Classifier i divides the data into the class belonging to class i and those not belonging to class i. The "one-against-one" method constructs classifiers for each of the class pairs, thus totally constructing $n(n-1)/2$ classifiers. Hsu *et al.* showed in [7] that "one-against-one" is more suitable for practical use. However, a recent publication [17] argued that "one-against-all" is as accurate as any other approaches. Since using "one-against-all" can save more training time, we adopt the "one-against-all" method in this paper.

6 Prediction Method

We now explain how our prediction method works. The prediction method actually consists of two parts, the training algorithm and the classification algorithm.

ALGORITHM 6.1. Training Algorithm.

```
1. Compute the feature matrix $M_f$;
2. Apply K-means($k$,$M_f$);
3. For clusters $c_1, \ldots, c_k$
4.     If ($Purity_i < \omega$)
5.         SVC_train($c_i$).
```

In the training algorithm (Algorithm 6.1), we first calculate the features of each sparse matrix and save them in a feature matrix M_f. Then we use the K-means algorithm to cluster all the objects (matrices) into k clusters according to their features. For each of the clusters, if its purity value is greater than a threshold value ω, most matrices in this cluster are considered to have the same solving status. If the purity value of a cluster is smaller than ω, we use the SVM classification algorithm to build training models for the cluster.

ALGORITHM 6.2. Classification Algorithm.

```
1. Given a matrix $A$, compute its features;
2. Find its nearest cluster $c_i$;
3. If( $Purity_i > \omega$)
4.     $SS_A = SS_{c_i}$
5. Else
6.     $SS_A$ = SVC_classify($c_i$,$A$).
```

Given a new matrix A, if we want to predict its solving status, we will apply the classification algorithm (Algorithm 6.2). First, the matrix features are computed. Then, the distance of the feature vector to the center of each cluster is calculated. The matrix is assigned to the cluster with the shortest distance. If the purity value of the cluster is greater than ω, the solving status of the matrix is defined as the solving status of the majority of the matrices in the cluster. Otherwise, the SVC training models of that cluster are used to classify the matrix to its corresponding solving status group.

After a certain number of new matrices have been inserted into each cluster, the training algorithm will be run again to reflect the changes brought by the newly added matrices.

7 Experiments and Results

We conduct some experiments to test the prediction method. The linear systems are constructed by using sparse matrices from MatrixMarket [15]. The right hand sides of the linear systems are constructed by assuming that the solutions are a vector of all ones. The initial guessed solutions are a vector of all zeros. The maximum number of iterations is 500. The convergence stopping criterion is that the 2-norm of the residual vectors is reduced by 7 orders of magnitude. The iterative method used is GMRES(20) and the preconditioners are matrix structure-based incomplete LU factorizations ILU0 and ILUK. We choose K to be 1, 2, 3 respectively and denote them as ILUK1, ILUK2 and ILUK3. We first predict whether a matrix can be solved by these preconditioned solvers. If we predict that a matrix cannot be solved by a particular preconditioner, we will also predict the reason(s) of the failure. Table 1 shows the possible solving status (SS) of running these preconditioned solvers and their meanings. If the prediction method encounters any problem during its processing, it will stop and return a corresponding status to indicate the problem. We use SVM^{Light} [8] for SVM classification.

Table 1: Solving status (SS) and the meanings.

SS	Meaning
-100	zero pivot in constructing a preconditioner
-5	zero row in constructing a preconditioner
-8	large condest, unstable preconditioner
-1	solver cannot converge in 500 iterations
0	successfully solved

7.1 Solvability prediction Here we compare the prediction results obtained by using the proposed prediction method (CSVC) and by using SVM classification (SVC) alone. A total of 66 matrix features are extracted and used. In the K-means algorithm, we choose k to be 12. ω is set to be 85%. Thus if more than 85% of the matrices in a cluster have the same SS, we will not do further classification on that cluster. In SVM classification, we apply the one-against-all method, constructing classifiers for each of the solving status and choosing the one with the highest value as a sparse matrix's final predicted solving status. The results are obtained by using a 5-fold cross validation.

Table 2: Predicted solving status related to ILU0.

SS	0	-100	-1	-8	Total
NM	143	132	32	12	319
NM_{SVC}	127	128	13	2	270
CT_{SVC}	88.8%	97.0%	40.6%	16.7%	84.6%
NM_{CSVC}	136	128	21	5	290
CT_{CSVC}	95.1%	97.0%	65.6%	41.7%	90.9%

The predicted solving status related to ILU0 are listed in Table 2. Here NM denotes the number of matrices, while CT denotes the correct rate (of successful prediction). Of all the 143 successfully solved matrices, we can correctly predict 136 of them by using CSVC, with the correct rate of 95.1%. On the other hand, if we use SVC, the correct rate is only 88.8%. If a matrix factorization encounters a zero pivot error, we can predict it with a correct rate of 97.0% with both SVC and CSVC. The accuracy is not high for predicting the cases of SS $= -1$ or SS $= -8$, the major reason for the poor predictions is that there are too few examples of such cases in the data sets. However, we can see the improvement of the prediction accuracy by using CSVC in stead of SVC. The correct rates are raised by 25% for both SS $= -1$ and SS $= -8$. Finally, the total correct rate for predicting ILU0 by SVC is 84.6%, while using CSVC it rises to 90.9%.

Table 3: Predicted solving status related to ILUK.

SS	0	-5	-1	-8	Total
ILUK1					
NM	197	74	37	11	319
NM_{SVC}	187	61	15	6	269
CT_{SVC}	94.9%	82.4%	40.5%	54.6%	84.3%
NM_{CSVC}	185	67	26	3	281
CT_{CSVC}	93.9%	90.5%	70.3%	27.3%	88.1%
ILUK2					
NM	198	69	34	18	319
NM_{SVC}	182	59	16	7	264
CT_{SVC}	91.9%	85.5%	47.1%	38.9%	82.8%
NM_{CSVC}	183	63	22	10	278
CT_{CSVC}	92.4%	91.3%	64.7%	55.6%	87.1%
ILUK3					
NM	203	69	31	16	319
NM_{SVC}	182	58	17	5	262
CT_{SVC}	89.7%	84.1%	54.8%	31.3%	82.1%
NM_{CSVC}	187	64	21	8	280
CT_{CSVC}	92.1%	92.8%	67.7%	50.0%	87.8%

We compare the accuracy in predicting the solving status of ILUK1, ILUK2 and ILUK3 in Table 3. For ILUK1, CSVC excels SVC in predicting SS $= -5$ and SS $= -1$, but its correct rate of predicting SS $= 0$ is slightly lower than that of SVC. It performs worse than SVC in predicting SS $= -8$, too. The total correct rate of CSVC is 88.1%, which is higher than that of SVC (84.3%). Compared with the results of ILUK1, CSVC works better for ILUK2 and ILUK3. It has higher accuracy in predicting every solving status. For ILUK2, it increases the CT by around 17% for SS $= -5$ and SS $= -1$. The total correct rate is raised from 82.8% to 87.1%. For ILUK3, CSVC exceeds SVC for every solving status, too. It gets 87.8% in total correct rate while SVC only gets 82.1%.

In conclusion, the prediction accuracy is improved in almost all the categories with CSVC for ILUK. Using CSVC, the correct rate of predicting whether a sparse matrix can be successfully solved is above 92% for all levels of the ILUKs. And the total correct rate is above 87%.

7.2 Choice of ω In the previous subsection, we choose ω to be 85% because it works best for our system, after a large number of experiments. Figure 2 shows the change of total correct rates of using CSVC for ILU0 and ILUKs with the increase of ω. We can see that the trend of the lines representing ILU0 and ILUKs are very similar. When ω is not large enough, with the increase of ω, CT increases. It means when the cluster is not very pure, choosing the solving status of the majority of the matrices as the solving status for every matrix will incur a large error. In this case, it is better to construct classification models for each of the solving status group in the cluster and predict the solving status of each matrix using these models. However, after ω reaches some point (85% in our experiments), the prediction error of the classification models is larger than not using these models. We do not need to construct classification models in this situation and can save training time.

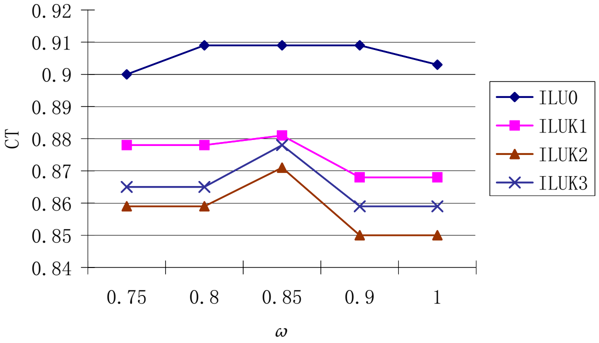


Figure 2: Relation of ω and the total correct rate.

Another point of Figure 2 is that even the lowest CT of ILU0 and ILUK in the figure is higher than the CT obtained by using SVC alone. It suggests that even if the ω value used is not the best one, it can also improve the prediction accuracy by using the CSVC method.

7.3 Cluster analysis Table 4 lists some statistics of the 12 clusters. P_{ilu0} denotes the purity of the cluster with respect to the solving status related to ILU0, and SS_{ilu0} denotes the solving status of the majority of the matrices in the cluster. Other columns for ILUKs have the similar meanings. There are 6 clusters with the purity value equal to 1 for both ILU0 and ILUKs. Matrices in Cluster 1 and Cluster 9 all have SS $= -100$ for ILU0 and SS $= -5$ for ILUKs. All matrices in Clusters 7, 10, 11, and 12 can be successfully solved. The last line of the table shows the average purity values for ILU0 and ILUKs. We can get:

$$\text{average}(P_{ilu0}) > \text{average}(P_{iluk3}) > \text{average}(P_{iluk2}) > \text{average}(P_{iluk1})$$

If we compare that with the improvement of total correct rate obtained by using CSVC instead of SVC, we can get a similar sequence:

$$\text{improvement}(CT_{ilu0}) > \text{improvement}(CT_{iluk3}) > \text{improvement}(CT_{iluk2}) > \text{improvement}(CT_{iluk1})$$

It seems that CSVC works better with higher average purity values. Our experiences in the experiments also show this trend. It suggests that if we use some clustering methods which can achieve higher purity value, we can gain higher total correct rate. We will do some experiments in the future to verify whether this hypothesis is true or not.

Table 5 describes the composition of the 6 purist clusters. It tells us which matrices have similar features. Matrices with the same name come from the same source. IMPCOL(3) means 3 matrices of IMPCOL type. If given a new matrix with the feature vector very close to one of these cluster centers, then we can predict that it will show similar property as the other matrices in the cluster when solved by some preconditioned solvers. We will investigate the features of these matrices and find out which features have large influence on deciding whether it can be solved by a preconditioner.

8 Concluding Remarks

The experimental results show that the prediction method gets promising results in predicting the solving status of sparse matrices by the matrix structure-based ILU type preconditioners. Using CSVC can improve the prediction accuracy over using SVC alone, the purest clusters generated can also give us some suggestions on which kind of matrices can get what results when solved by these preconditioned solvers.

This implementation and the reported experimental results are the first indication and evaluation of our study in using data mining techniques in large scale scientific computing applications. We will study and experiment the prediction method on more sophisticated preconditioners like ILUT and use feature selection method to reduce the computational cost in our future work.

References

[1] M. Benzi. Preconditioning techniques for large linear systems: a survey. *J. Comput. Phy.*, **182**, 418-477, 2002.

[2] C. Burges. *A tutorial on support vector machine for pattern recognition.* Kluwer Academic Publishers, 1998.

[3] C. Campbell. Kernel methods: A survey of current techniques. *Neurocomputing.*, 48 (2002) 63-84.

[4] T. Davis. University of Florida sparse matrix collection. *NA Digest*, 97(23), June 7, 1997.

Table 4: Cluster statistics.

Clusters	NM	P_{ilu0}	SS_{ilu0}	P_{iluk1}	SS_{iluk1}	P_{iluk2}	SS_{iluk2}	P_{iluk3}	SS_{iluk3}
Cluster 1	11	1.0	-100	1.0	-5	1.0	-5	1.0	-5
Cluster 2	72	0.69	-100	0.48	0	0.50	0	0.56	0
Cluster 3	26	0.88	-100	0.62	0	0.69	0	0.73	0
Cluster 4	27	0.37	-100	0.26	-1	0.30	-1	0.37	-1
Cluster 5	16	0.69	0	0.75	0	0.81	0	0.81	0
Cluster 6	18	0.78	0	0.83	0	0.83	0	0.83	0
Cluster 7	5	1.0	0	1.0	0	1.0	0	1.0	0
Cluster 8	50	0.92	0	0.90	0	0.86	0	0.86	0
Cluster 9	31	1.0	-100	1.0	-5	1.0	-5	1.0	-5
Cluster 10	41	1.0	0	1.0	0	1.0	0	1.0	0
Cluster 11	7	1.0	0	1.0	0	1.0	0	1.0	0
Cluster 12	15	1.0	0	1.0	0	1.0	0	1.0	0
Average	26.6	0.86	-	0.82	-	0.83	-	0.85	-

Table 5: Matrix composition of the pure clusters.

Clusters	Matrices
Cluster 1	IMPCOL(3), QH(2), RW(2), TOLS(2), WEST0132, ZENIOS
Cluster 7	BCSSTM(5)
Cluster 9	BP(9), GEMAT(2), IMPCOL(2), PSMIGR_2, SHL(3), STR(4), WEST(10)
Cluster 10	FS(4), STEAM(2), BUS(4), BCSSTK(26), BCSSTM(3), GR_30_30, LUND_B
Cluster 11	BCSSTM(7)
Cluster 12	FS_183_6, MCCA, MCFE, PORES_1, SAYLR1, STEAM3, BCSSTK(3), LUND_A, NOS(5)

[5] I. S. Dhillon, D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143-175, 2001.

[6] E. N. Houstis, J. R. Rice, and R. Vichnevetsky. Intelligent mathematical software systems. In *Proceedings of the 1st IMACS/IFAC International Conference on Expert Systems for Numerical Computing*, New York, 1990. North-Holland.

[7] C. W. Hsu and C. J. Lin. A comparison of methods for multi-class support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2001

[8] T. Joachims. Making large-scale SVM learning practical. *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges and A. Smola (ed.), MIT-Press, 1999.

[9] Y. Li, S. Gong, H. Liddell. Support vector regression and classification based multiview face detection and recognition. In *Proc. of the IEEE International Conference on Automatic Face and Gesture Recognition (FGR'00).*, Grenoble, France, 2000.

[10] N. Kang, E. S. Carlson, J. Zhang, Parallel simulation of anisotropic diffusion with human brain DT-MRI data. *Computers and Structures*, 82(28):2389–2399, 2004.

[11] S. Karaa, C. C. Douglas, J. Zhang, Preconditioned multigrid simulation of an axisymmetric laminar diffusion flame. *Math. Comput. Model.*, **38**, 269-279, 2003.

[12] J. Lee, C. Lu, J. Zhang, Performance of preconditioned Krylov iterative methods for solving hybrid integral equations in electromagnetics. *J. of Applied Comput. Electromagnetics Society*, **18**, 54-61, 2003.

[13] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Symposium on Math, Statistics, and Probability.*, Univ. of California Press, p. 281-297, 1967.

[14] J. A. Meijerink, H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comput.*, 31:148–162, 1977.

[15] http://math.nist.gov/MatrixMarket.

[16] J. Platt, N. Cristianini and J. Shawe-Taylor. Large margin DAGS for multiclass classification. *Advances in Neural Information Processing Systems*, 12 ed. S.A. Solla, T.K. Leen and K.-R. Muller, MIT Press, 2000.

[17] R. Rifkin, A. Klautau In Defense of One-Vs-All Classification. *Journal of Machine Learning Research*, 5 (2004) p. 101-141.

[18] Y. Saad. ILUT: a dual threshold incomplete LU preconditioner. *Numer. Linear Algebra Appl.*, 1(4):387–402, 1994.

[19] Y. Saad, *Iterative Methods for Sparse Linear Systems.* PWS, New York, 1996.

[20] V. N. Vapnik. *Statistical Learning Theory.* John Wiley and Sons, New York, 1998.

[21] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[22] V. Vural, J. G. Dy. A hierarchical method for multi-class support vector machines. In *Proc. of the Twenty-first International Conference on Machine Learning.*, Banff, Alberta, Canada, July 2004

[23] S. Xu, E. Lee, J. Zhang. An interim analysis report on preconditioners and matrices. Technical Report No. 388-03, Department of Computer Science, University of Kentucky, Lexington, KY, 2003.

[24] S. Xu, E. Lee, J. Zhang. Designing and building an intelligent preconditioner recommendation system (a progress report). In *Abstracts of the 2003 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications*, Napa, CA, 2003.

[25] J. Zhang. Preconditioned Krylov subspace methods for solving nonsymmetric matrices from CFD applications. *Comput. Methods Appl. Mech. Engrg.*, 189(3):825–840, 2000.

[26] J. Zhang. Performance of ILU preconditioners for stationary 3D Navier-Stokes simulation and the matrix mining project. In *Proceedings of the 2001 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications*, p. 89–90, Tahoe City, CA, 2001.

Better Bond Angles in the Protein Data Bank

C.J. Robinson and D.B. Skillicorn
School of Computing
Queen's University
{robinson,skill}@cs.queensu.ca

Abstract

The Protein Data Bank (PDB) contains, at least implicitly, empirical information about the bond angles between pairs of amino acids. There is considerable variation in the observed values for a given amino acid pair, and it is not clear whether this variation represents a wide range of conformal possibilities or is due to noise. We show, by applying singular value decompositions to sets of examples of particular amino acid sequences, that there appear to be relatively few possible conformations for a given amino acid pair, and hence noise is a plausible explanation for the variation in the raw data. This has implications for secondary structure prediction which typically depends on the PDB values.

1 Introduction

Proteins are constructed from long chains of 20 unique amino acid building blocks. Every amino acid shares a similar chemical structure but with a distinguishing chemical component, a side chain. The backbone of a protein is created by linking these similar structures together. Each pair of amino acids along the backbone is joined in a way that is conventionally described by two angles of rotation; ϕ and ψ. These bond angles are constrained by numerous influences, but are assumed to be locally dominated by the physical and chemical properties of the side chains. A common visualization of possible bond angles is to create a plot with ϕ angles on the horizontal axis and ψ angles on the vertical axis. This is known as a Ramachandran Plot; and areas of this graph correspond to common structural formations at the secondary structure level (Figure 1).

The Protein Data Bank (PDB) is a repository of protein structure, mainly gathered using X-ray Crystallography (XC) and Nuclear Magnetic Resonance (NMR). Implicit in the PDB is the conformation of each protein's backbone sequence. The actual bond angles associated with a given pair of amino acids in the PDB show wide variation. In some cases, it is clear that this is due to different conformations; the bond angle values are far apart. In other cases it is not clear whether the variations in values correspond to conformal possibilities or simply to noise. We address this question empirically and show that it is plausible, perhaps likely, that most of the variation in values is due to noise. As a result, we are able to show distinct conformations of amino acid pairs that are not distinguishable in the raw data. By extending these results to larger sequences of amino acids, we suggest that secondary structure can be built from primary structure in a bottom-up fashion. As a side-effect we are able to produce Ramachandran plots with much more tightly constrained admissible regions than in conventional plots.

Our strategy is to select all occurrences of a particular amino acid sequence from the PDB, together with the bond angles between its members. For example a sequence A-B-C could be selected where A,B and C are amino acid types and each dash corresponds to a ϕ,ψ bond angle pair. Singular Value Decomposition (SVD) is then applied to the resulting dataset; the resulting matrices are truncated at some k components; and then remultiplied to produce a matrix analogous to the original bond angles. This new matrix can be thought of as defining *canonical bond angles* for the amino acid sequence being considered. The different occurrences show strong tendencies to cluster in the transformed space, providing evidence of a limited set of conformations distorted by noise rather than a wide range of possible conformations. Since most secondary structure prediction uses data from the PDB, much improved results would be expected from using canonical, rather than measured, bond angles.

2 Related Work

The effects of neighboring amino acids, *n-1*, *n+1*, on the conformation of amino acid n was studied as early as the 1970's. Advanced, automated analysis of this variety is now possible over the WWW via the Conformation Angles Database (CADB) [9]. Interfaces to CADB can automatically generate Ramachandran plots of specific amino acids with respect to neighboring residues. This is currently limited to three amino acids and is also restricted by the nature of information contained in the CADB.

Aggregating PDB data to create empirical Ramachandran plots (as opposed to ones based on energy minima, see Section 4.1) has been done by Kleywegt and Jones [5] and later by Hovmoller *et al.* [3]. The plots were created by extracting bond angles for every instance of an amino acid within the PDB. These types of analysis are useful in that they utilize all the information available to provide a detailed plot, but the relative simplicity limits the application of the results.

Clustering algorithms have been applied to length 5 amino acid sequences obtained from the PDB [6]. The clustering was done based on amino acid type and not related to bond angles or sequence conformation. Though similar in concept to our work, the results are fundamentally different as the clustering is not based on bond angles.

A geometric analysis of bond angles for a specific protein structure is available directly at the PDB website. The analysis can show bond angles that deviate from the generally accepted values by more than a given threshold. This is a generally accepted method to identify potential noise and/or errors introduced from the XC or NMR process [1]. Since it is a simple comparison of threshold values, the results obtained do not reflect great confidence and the analysis is limited to one protein structure at a time.

A recent paper by Rost *et al.* [8] summarizes work on secondary structure prediction. Most secondary structure prediction methods are based on information derived from conformation data implicit in the PDB.

3 Matrix Decompositions

The singular value decomposition of a matrix A is

$$A = USV'$$

where the dash indicates the transpose. If A is $n \times m$ and has rank r, then U is $n \times r$, S is an $r \times r$ diagonal matrix with decreasing entries $\sigma_1, \sigma_2, \ldots, \sigma_r$ (the singular values), and V is $r \times m$. In addition, both U and V are orthogonal, so that $UU' = I$ and $VV' = I$. In most practical datasets, $r = m$.

The most useful property of SVD for our purposes is that the transformation captures as much variation in the original data as possible in the first transformed dimension, as much as possible of what remains in the second dimension, and so on. Hence, if we truncate the decomposition so that U is $n \times k$, S is $k \times k$, and V is $k \times m$, for some small k, then we have discarded dimensions that have little influence on the correlational structure of A. The dimensions from $k+1$ to r can be considered to represent noise in the original data.

Remultiplying the truncated matrices produces a new matrix that has the same shape and interpretation as A but has been 'denoised'. The truncation parameter, k, should be chosen so that significant information is retained but insignificant discarded. The magnitudes of the singular values are a measure of how important each dimension of the decomposition is. Plotting the values of the diagonal of S and choosing k at the earliest point where these values become small is often a good selection mechanism.

An alternate interpretation of the transformed space produced by an SVD is that points are placed close to other points with which they are correlated. Hence if the original data describes objects of a few different kinds, distorted by noise, we would expect to see tight clusters in the transformed space (which can be plotted and visualized directly if $k = 2$ or 3). On the other hand, if the original data describes data without much similarity, we would not expect to see clusters in the transformed space in any dimensions. SVD reveals the latent cluster structure of data.

4 Background and Methods

This section describes current methods of obtaining protein structure and the inherent problems, databases of acquired protein structure, and the limitations of applying SVD and other data mining algorithms to such data.

4.1 Obtaining structure Current methods for determining the 3-dimensional structure of a protein are slow, error-prone and expensive. The readily available methods employed today are X-ray crystallography (XC) and Nuclear Magnetic Resonance (NMR); both are physical processes. XC and NMR involve determining the 3-dimensional coordinates of every atom in a protein within a known error range. Data collected from both methods are not directly converted to atomic co-ordinates but require human input, interaction and heuristics to refine the data. This is a further possible source of error [4]. The structure of about 28,000 proteins has been determined using these methods but millions of proteins are known to exist.

The Protein Data Bank (PDB) is the worldwide depository for protein structure, almost all of which have been obtained from XC and NMR. The format for protein structure has been carefully designed to be as flexible as possible allowing the data to be utilized in many different fields of study [1]. The PDB consists of individual files for each protein entry, and contains the atomic co-ordinates of atoms within the protein, but does not directly contain bond angles for amino acids in the primary sequence. Unfortunately, data mining applications are not natively supported by this standard format of the PDB.

4.2 Database formats There are many derivatives of the PDB, only a few of which contain bond angles of amino acids in an easily accessible format. The Conformation Angles Database (CADB) [9] and Dihedral Angle Database (DAB) [2] are the newest and most comprehensive sources. CADB is a self-limited database which excludes proteins with homologous sequences up to a certain threshold. For data mining purposes, it would be beneficial not to exclude any data *a priori*. CADB also will only supply bond angles for small sequences of amino acids. DAB contains bond angles for every possible set of length 2, 3, 4 and 5 amino acid sequences. However, the database is not currently publicly available. Both of these databases lack the ability to supply a set of bond angles in a matrix format based on a simple query (i.e. for a specific sequence of amino acids, or for every set of 8 amino acids). Since most data mining applications, and in particular SVD, require data to be in matrix form, a new database was required.

4.3 Ramachandran plots Ramachandran plots are a way to display the possible conformation of an amino acid pair by plotting the admissible regions based on energy considerations. Figure 1 shows a typical plot. The region near the top left corresponds to conformations associated with a β sheet, with the two peaks corresponding to parallel and anti-parallel secondary structure. The region midway down and to the left corresponds to conformations associated with α helices. The third region corresponds to conformations of anticlockwise α helices. Not all amino acids are constrained in this way; for example pairs involving glycine can exhibit a much larger range of conformations. We will use Ramachandran plots to compare the conformations possible in the raw PDB data and those suggested after denoising with SVD.

4.4 Methods Every protein file listed in the Protein Data Bank was downloaded, converted and appended to a datafile. Bond angles were computed from atomic coordinates for each pair of amino acids in the primary sequence using TORSIONS [7]. The output of TORSIONS was appended to a datafile in the format: ..., amino-acid(location), ϕ, ψ,... (e.g. ..., ALA(150), -150.595, -63.539, SER(151), ...), where each line is the structure of one protein. Due to the

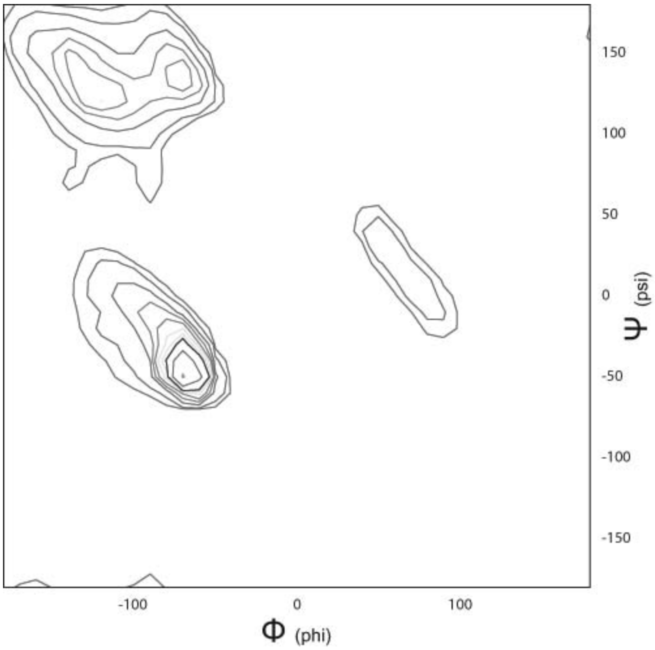


Figure 1: Typical Ramachandran plot (from Kleywegt and Jones (1996) [5]).

time-intensive nature of compiling a database into this format, this intermediate, flexible format was chosen for ease of access for other possible studies. The results presented here are based on 27,544 files which were downloaded from the PDB on November 16, 2004 and transformed into the new format. From this new derivative of the PDB, 125 sets of length 3 sequences, 625 sets of length 4 sequences and 625 sets of length 5 sequences were extracted for analysis.

The natively implemented svd function in Matlab 6.5r13 was used to perform the decomposition on small datasets. For larger datasets JAMA v5 was used with Java 1.4.2-04 to perform SVD. The resultant decomposed data was geometrically interpreted by plotting the first three dimensions of the U matrix.

The following steps were performed:

- Given an amino acid sequence of length m, the $2m-2$ internal bond angles associated with each occurrence of this sequence were extracted from the PDB. There are typically 1000–5000 examples of an amino acid sequence of length 3 in the PDB, 100–500 examples of sequences of length 4, and 0–200 examples of sequences of length 5.
- An SVD was performed on the resulting matrix whose rows correspond to examples of the given amino acid sequence and whose columns correspond to a bond angle at a particular position in the sequence. For example a length 4 amino acid sequence (A-B-C-D) would have the following matrix row for each instance: ϕ_{AB} ,ψ_{AB} ,ϕ_{BC} ,ψ_{BC} ,ϕ_{CD} ,ψ_{CD} .
- The resulting decomposition was truncated at $k = 3$, a value chosen after inspection of a large number of plots of singular values. The first 3 columns of the U matrix was plotted for visualization.
- The truncated matrices were remultiplied to give a matrix of canonical bond angles.
- Each cluster in the transformed space was fitted with a 3-dimensional ellipse and the ellipse mapped back into bond angle space using the SVD ‘in reverse’. This ellipse defines a region of the Ramachandran plot corresponding to the cluster.

5 Results

There are many possible sequences of length 3, 4, and 5, so we only show some typical results here.

We begin by considering the sequence LEU-VAL-ARG of length 3. There are 4529 examples of this sequence in the dataset. Figures 2 and 3 show histograms of the bond angles of the LEU-VAL bond taken from these examples. There is obvious structure in both histograms, but it is hard to make use of it. The distribution of ψ angles suggests two clusters; but is the distribution of ϕ angles one big and one small cluster, or two big clusters and one small one? It is not straightforward to build conformations from such information. It is also not clear how much of the visible variation is due to noise and how much represents different possible conformations. Figure 4 is a Ramachandran plot of the bond angles for the LEU-VAL bond. It is clear

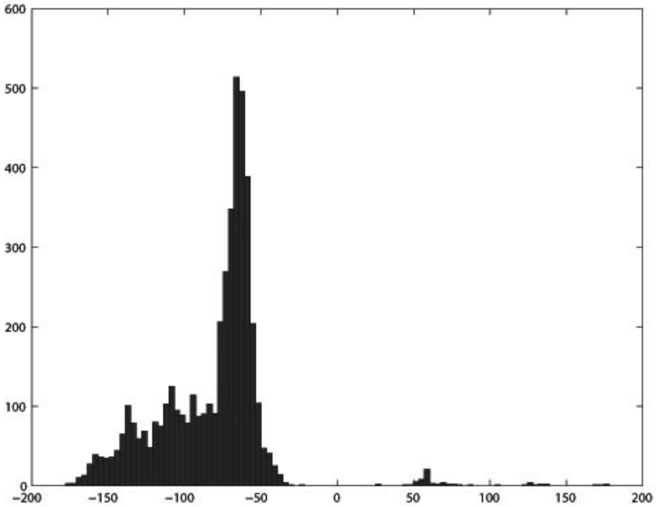


Figure 2: Histogram of the ϕ angles of the LEU-VAL bond in the sequence LEU-VAL-ARG.

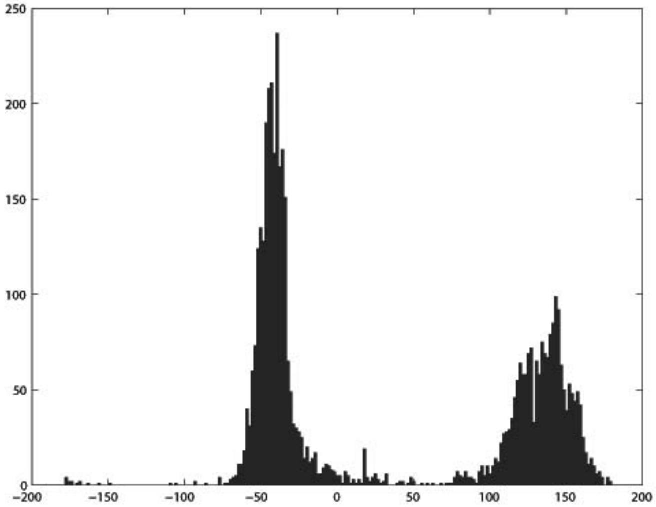


Figure 3: Histogram of the ψ angles of the LEU-VAL bond in the sequence LEU-VAL-ARG.

that there are some α helix conformations, some β sheet conformations, and a few anticlockwise α helix conformations, but the space of possible conformations is not very constrained.

Figure 5 shows the first 3 dimensions of the U matrix obtained from an SVD of the matrix of bond angles for the sequence LEU-VAL-ARG.

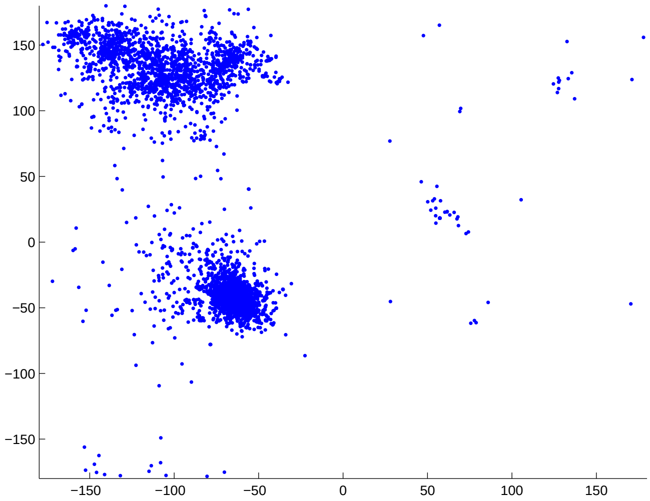


Figure 4: Ramachandran plot for the LEU-VAL bond in the sequence LEU-VAL-ARG.

There are four clusters visible, two large and two much smaller. There are a number of outliers, and perhaps some smaller clusters, but they represent a small fraction of the total number of examples.

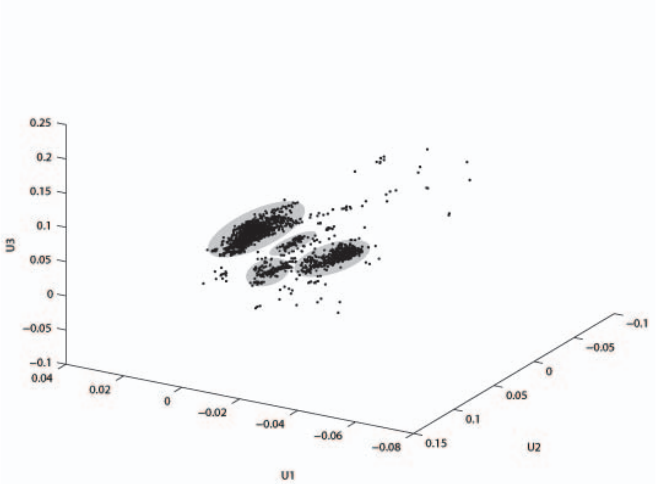


Figure 5: 3-dimensional plot of SVD transformed space for the amino acid sequence LEU-VAL-ARG.

Figure 6 shows the location of these clusters when mapped back into bond angle space for each of the pair bond angles. There are multiple possible

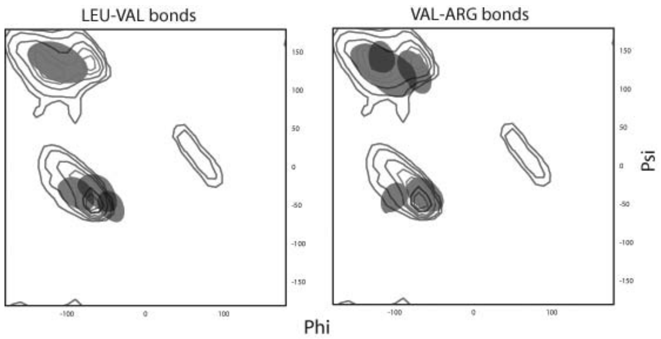


Figure 6: A Ramachandran plot overlayed with regions obtained from mapping clusters from SVD transformed space back to bond angle space for the amino acid sequence LEU-VAL-ARG. Each region is labelled in gray, darker where regions overlap. Compare with Figure 4

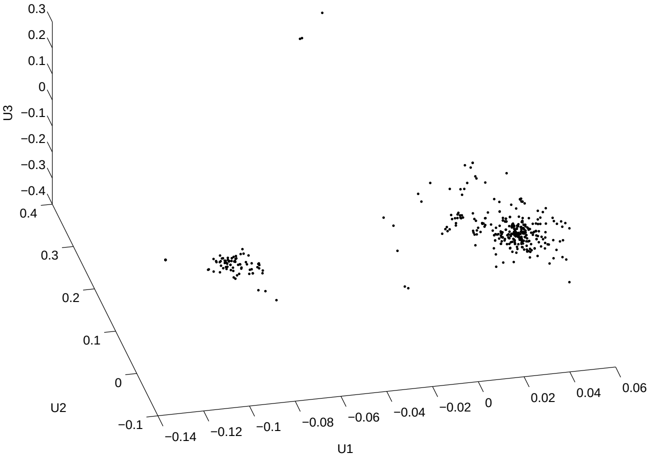


Figure 7: 3-dimensional plot of SVD transformed space for the amino acid sequence LEU-VAL-ARG-ILE.

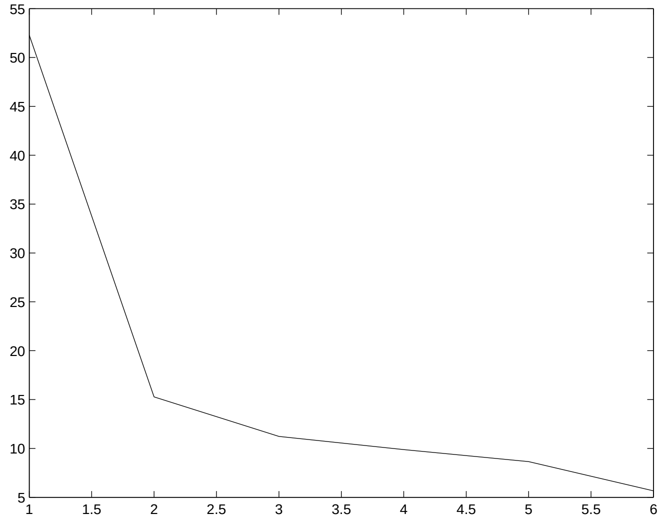


Figure 8: Plot of the singular values of SVD for the amino acid sequence LEU-VAL-ARG-ILE.

conformations for each of the bond angles: the LEU-VAL bond exists in 3 α helix conformations (with different pitches) and 1 β sheet conformation. The conformations for the VAL-ARG bond split more finely, with 2 α helix conformations, and 3 β sheet conformations, two of which correspond to the same cluster in the transformed space. In other words, 2 of the conformations of the overall sequence are α helices, 1 is a β sheet, and one exhibits transitions from one shape to another at the VAL amino acid, with two different possible β sheet conformations. Note that these transitions are abrupt; there is little evidence that the 'end'of the α helix changes shape because of the conformation of the adjacent bond.

Figure 7 shows the equivalent SVD transformed space for a sequence that extends the one we have just been considering: LEU-VAL-ARG-ILE. There are 417 examples of this sequence in the dataset. We see that there are two well-separated clusters with hardly any outliers. Figure 8 shows the singular values for the matrix of examples of this amino acid sequence. Almost all of the variation is in the first two dimensions.

Figure 9 shows the Ramachandran plot derived from these two clusters. As expected, the plots of each of the bond shows well-separated conformal possibilities, an α helix and a β sheet; and the conformations for the first two bonds are subsets of those computed from the sequence of length three. It seems likely that each of the possibilities for the bonds match, so that there are only two conformations for this entire length four sequence (and notice the tightening towards the right hand end for both conformations). We have looked at many sequences of length 3 and 4, and the results are similar.

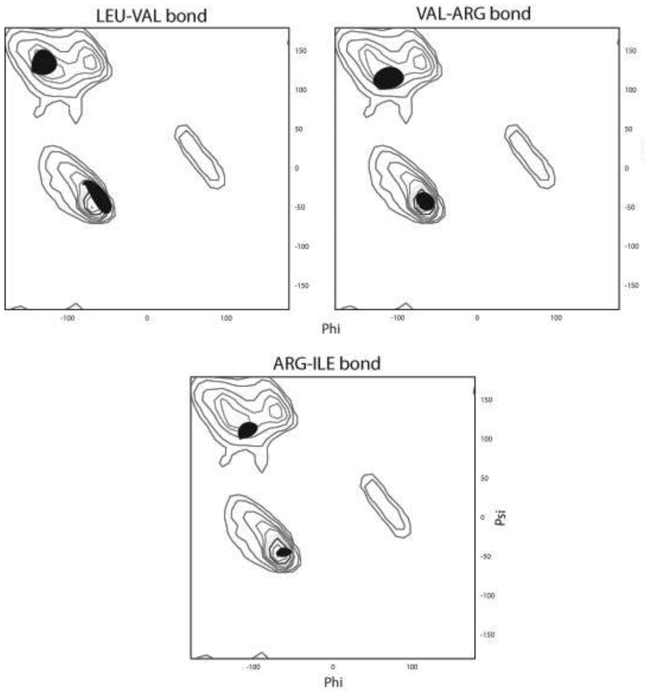


Figure 9: A Ramachandran plot overlayed with areas obtained from mapping cluster from SVD transformed space back to bond angle space for the amino acid sequence LEU-VAL-ARG-ILE.

These results suggest that a much improved version of the PDB's bond angles could be produced by systematically denoising the existing data. However, this is difficult in practice. There are a large number of amino acid sequences of short length, so a great deal of computation is required even to apply the techniques described here to all of their bond angles. It then remains to check whether the canonical bond angles derived from sequences of length 3 agree with (or can be fitted to) those obtained from sequences of length 4 for the same amino acid pair. However, this suggests the possibility of a dynamic programming style algorithm for determining conformations for a sequence of length l from the conformations of overlapping sequences of length $l-1$. We are exploring this possibility further.

5.1 Larger structure

A random sampling of 1,557,072 length 5 amino acid sequences (approximately a 10% sample of all possible length 5 sequences) was extracted from the PDB. Figure 10 shows the 3-dimensional plot of the SVD transformed space that results. The overall structure is a diamond of clusters. The leftmost cluster corresponds to sequences whose basic conformation is straight (i.e. they are part of β sheets) while the rightmost cluster corresponds to α-helices for the entire sequence. The clusters along the edge of the diamond appear to be sequences in transition between these two basic conformations. For example, the first amino acid in a sequence can be part of an α helix, while the remaining amino acids form a straight segment. The intermediate clusters capture these different conformation possibilities (each conformation appears as two different clusters because it depends on which end we consider first). Clusters in the middle appear to capture conformations with more than one transition, for example from α helix to straight segment to α helix. This figure shows how an SVD analysis of bond angle data can help to elucidate average structure in the PDB. These experiments have been repeated for longer amino acid chains with similar results. It is clear from these results that the transitions between helices and sheets are typically quite abrupt – the pitch of a helix does not appreciably alter as it transitions to a straight segment.

6 Conclusions

We have applied singular value decomposition to datasets of bond angles for particular short sequences of amino acids, using the values from the PDB. The raw data contains a great deal of variability, and it is not clear to what extent this represents noise (or errors) or different conformal possibilities. By examining the clustering structure in the space to which SVD transforms the data, we have provided some evidence that the main source of variation in the raw data is noise. Relatively few conformal possibilities are revealed in the transformed space.

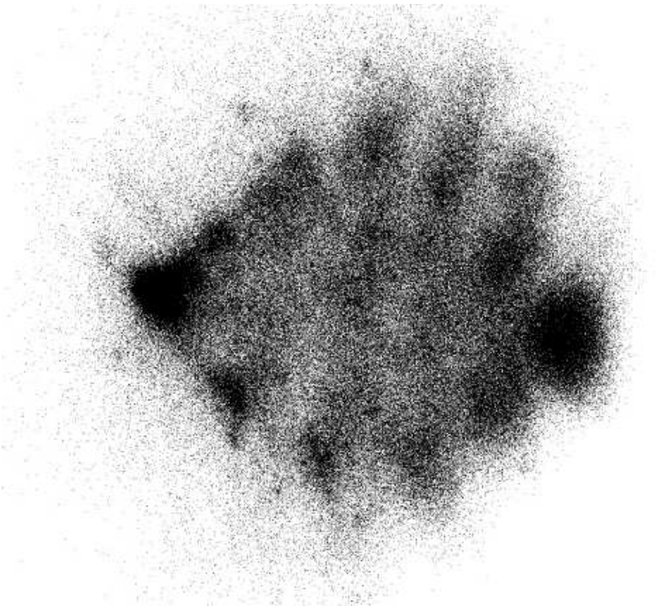


Figure 10: 3-dimensional plot SVD transformed space for a large set of arbitrary amino acid sequences of length 5.

Mapping the clusters back to the original bond angle space produces a version of each dataset containing 'canonical' bond angles, that is values that have been denoised. These bond angles are constrained to much smaller regions of Ramachandran plots, and exhibit coherence when the same cluster is followed along a sequence of bond angle pairs.

Many secondary structure prediction algorithms and functional studies are based on the PDB. Our results suggest that some effort should be spent on denoising the data before drawing conclusions about more complex structure from it. The techniques described here cannot be applied directly to the entire PDB. Although a single SVD on the entire PDB is (just) possible, new entries are being added all the time, and it is not a computation that is attractive to repeat regularly at this time. Replacement of bond angle data piecemeal using our techniques on sequences of medium length should be straightforward, but requires further analysis of the variation in results for an amino acid pair considered as part of a sequence of length 3, of length 4, and so on.

The elicitation of more robust conformations for short amino acid sequences suggests a method for discovering the conformations of longer sequences by assembling the short conformations into longer ones. We are pursuing this direction.

References

[1] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The Protein Data Bank. *Nucleic Acids Research*, pages 235–242, 200.

[2] S. Dayalan, S. Bevinakoppa, and H. Schröder. A dihedral angle database of short sub-sequences for protein structure prediction. In *2nd Asia-Pacific Bioinformatics Conference*, Dunedin, New Zealand, 2004. Conferences in Research and Practice in Information Technology.

[3] S. Hovmoller, T. Zhou, and T. Ohlson. Conformations of amino acids in proteins. *Biological Crystallography*, D58:768–776, 2002.

[4] G. Kleywegt. Validation of protein crystal structures. *Biological Crystallography*, D56:249–265, 2000.

[5] G. Kleywegt and T. Jones. Phi/psi-chology: Ramachandran revisited. *Structure*, 4:1395–1400, 1996.

[6] W. Li, L. Jaroszewski, and A. Godzik. Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics*, 17:282–283, 2001.

[7] A. Martin. Torsions. acrmwww.biochem.ucl.ac.uk/programs/torsions/, 2004.

[8] B. Rost. Review: Protein secondary structure prediction continues to rise. *Journal Structural Biology*, 134:204,218, 2001.

[9] S. Sheik, P. Ananthalakshmi, G. Ramya Bhargavi, and K. Sekar. Cadb: Conformation angles database of proteins. *Nucleic Acids Research*, 31:448–451, 2003.